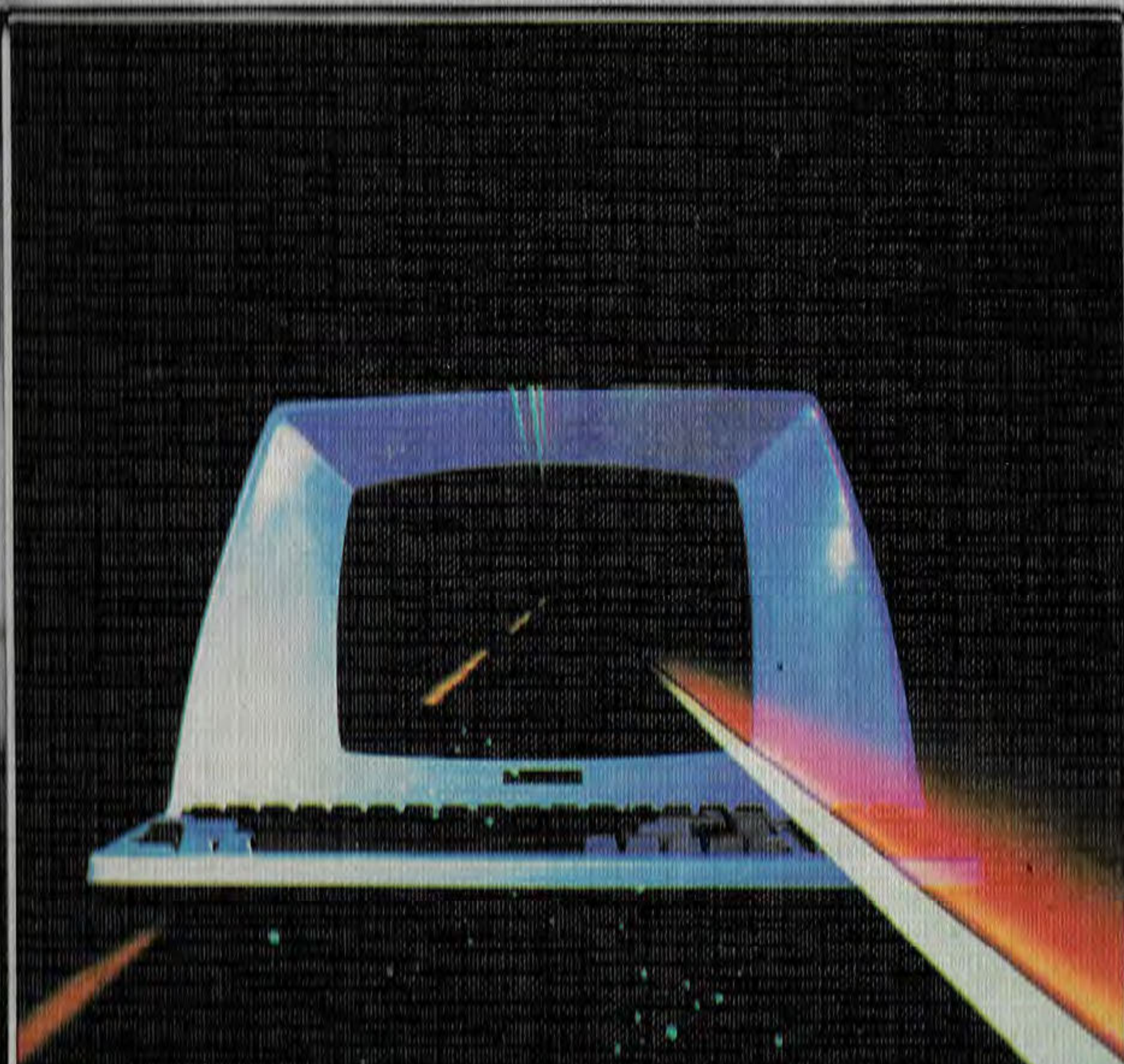
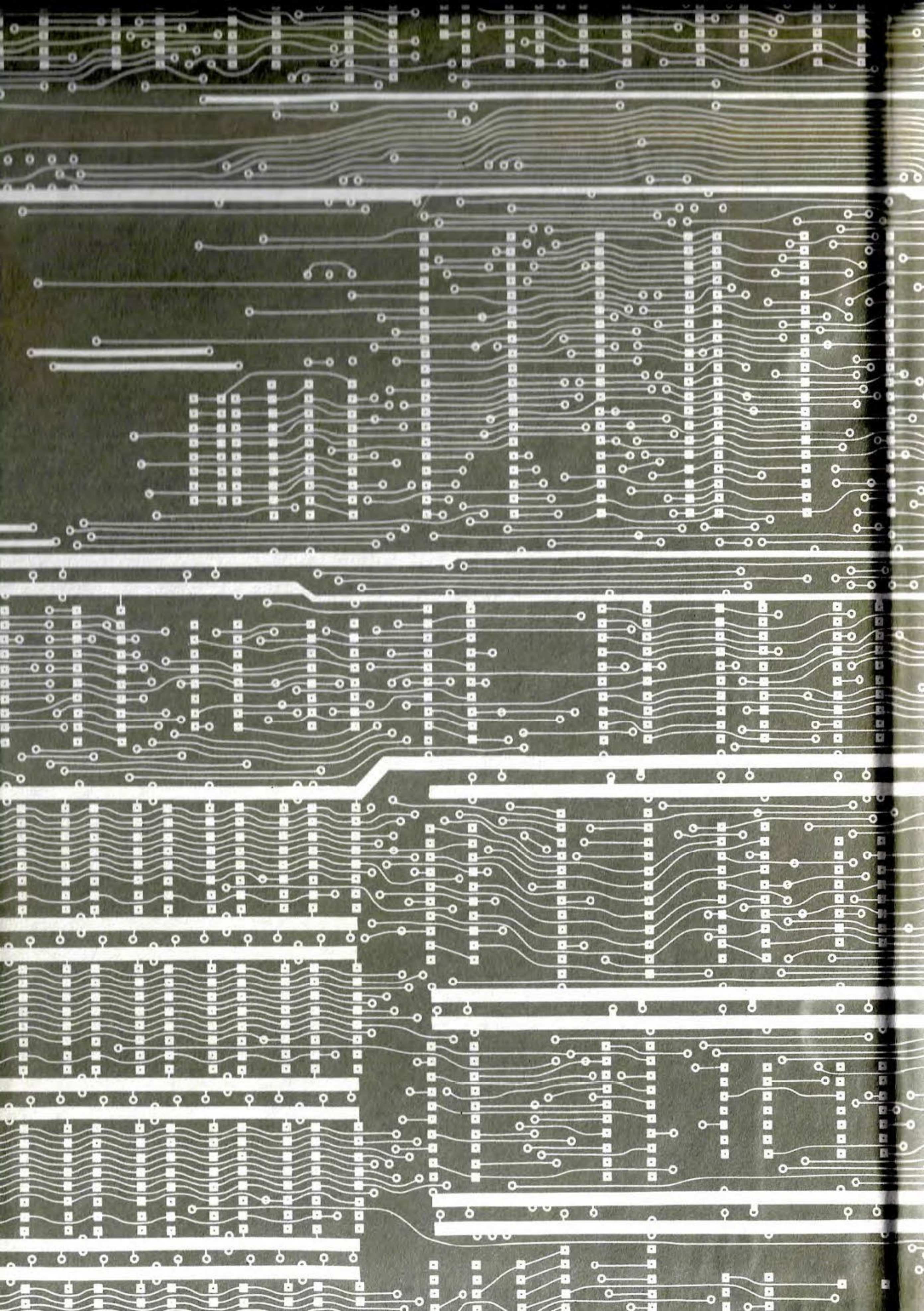
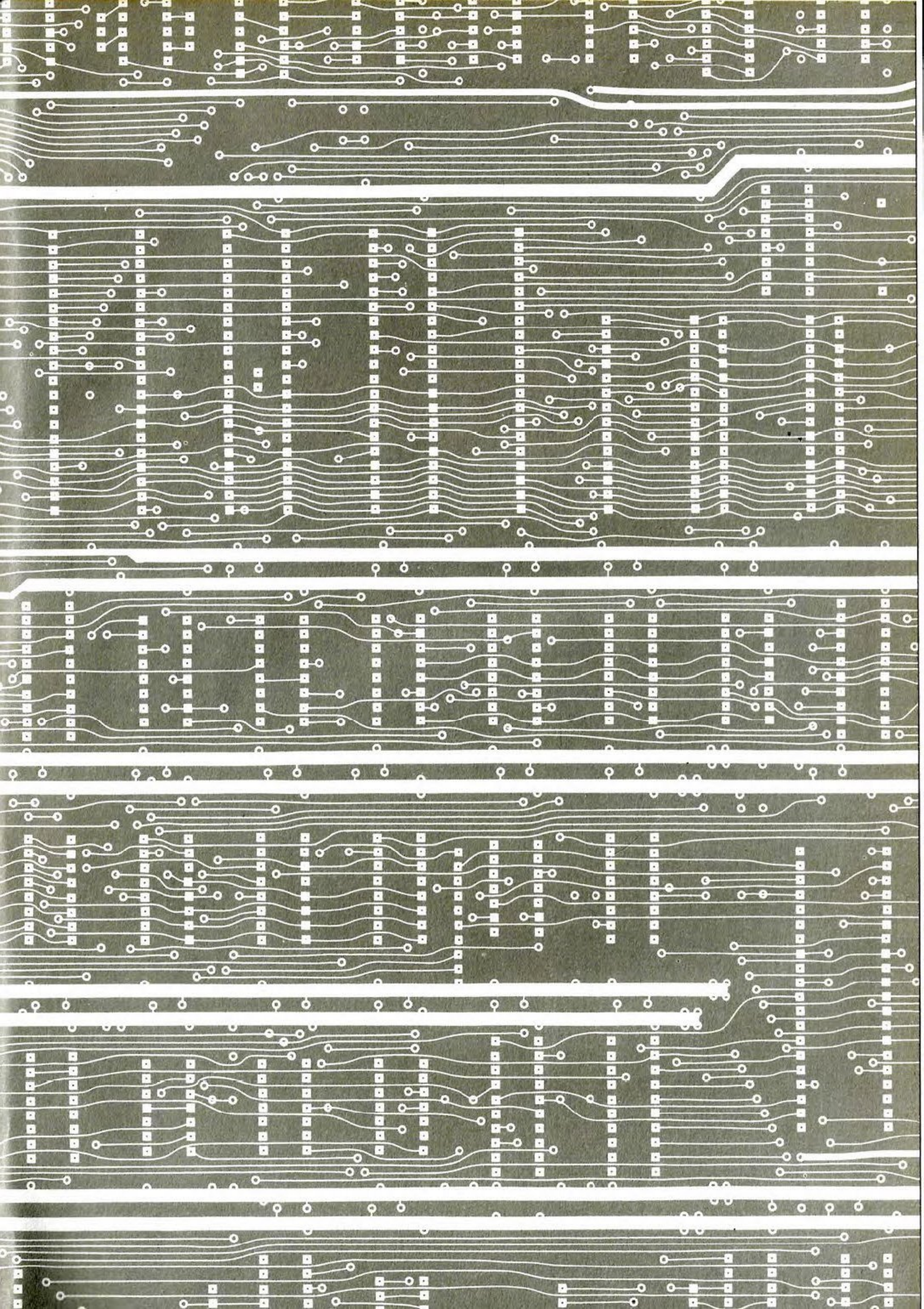
BASIC

ENCICLOPEDIA DE LA INFORMATICA
MINIORDENADORES Y ORDENADORES PERSONALES



EDICIONES FORUM





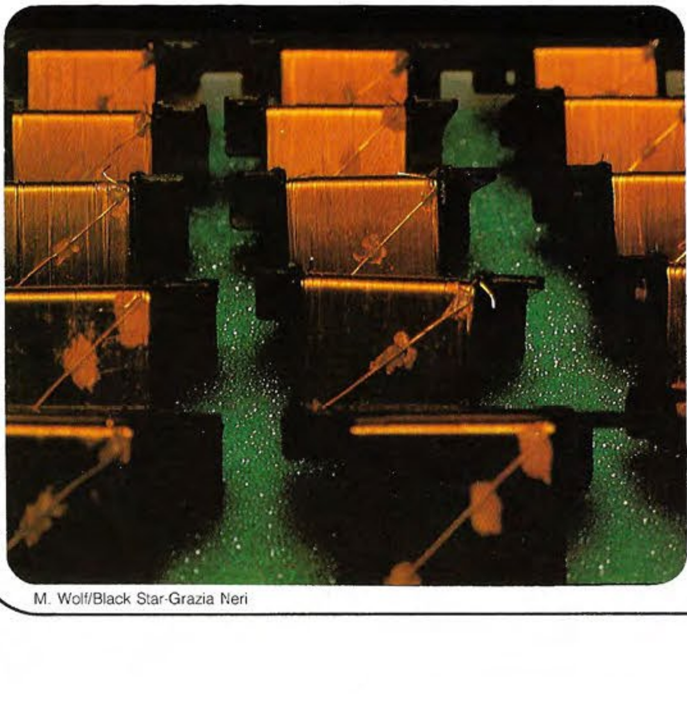
BASIC

ENCICLOPEDIA DE LA INFORMATICA.
MINIORDENADORES Y ORDENADORES PERSONALES

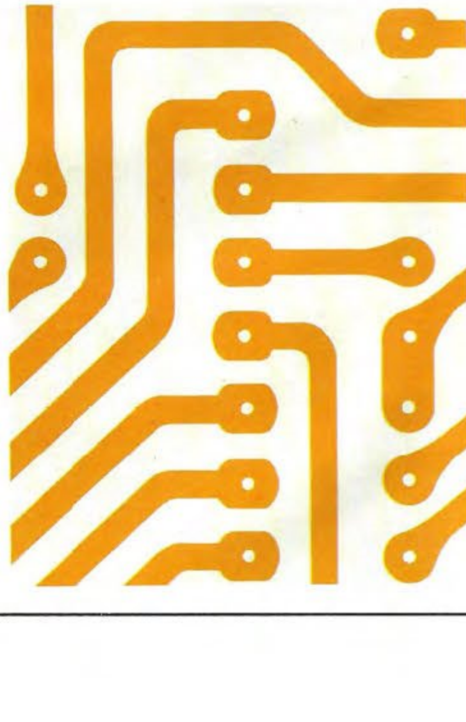
ENCICLOPEDIA DE LA INFORMATICA. MINIORDENADORES Y ORDENADORES PERSONALES



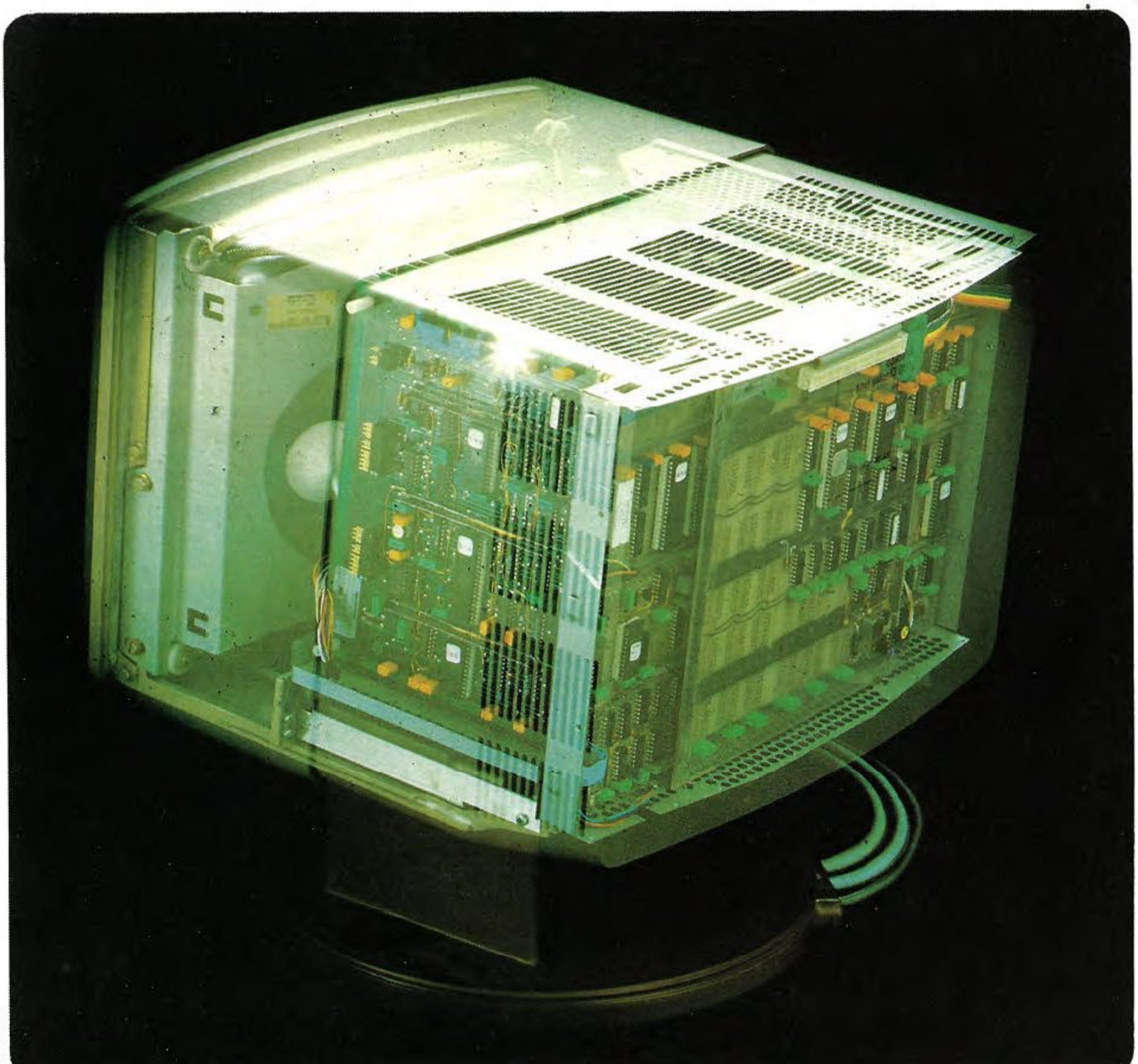
Globe Photos-Marka



M. Wolf/Black Star-Grazia Neri



BASIC 4



Ericsson

BASIC

ENCICLOPEDIA DE LA INFORMATICA. MINIORDENADORES Y ORDENADORES PERSONALES

Presidente
José Manuel Lara

Director Ejecutivo
Jesús Domingo

Dirección editorial
R.B.A., Proyectos Editoriales, S.A.

Dirección técnica
Sante Senni

Con el asesoramiento de la Sociedad **E.G.S.**

REDACCION

Dirección editorial: Gabriella Costarelli
Redactor jefe: Marcella Marcaccini
Secretaria de redacción: Giulia Abriani, Giovanna Aloisi
Revisión: Ugo Spezia
Corrección: Maria Albergo, Laura Salvini, Graziella Tassi
Recopilación material gráfico: Carla Bertini, Rossella Pozza
Producción: Piergiorgio Palma
Secretaria de producción: Maria Rita Ciucci
Diseño y dirección artística: Vittorio Antinori
Jefe estudio gráfico: Roberto Sed
Compaginación: Alberto Berni, Riccardo Catani, Patrizia Fazio
Dibujos: Renato Lazzarini, Gianni Mazzoleni, Rolando Mazzoni
Traducción: Carlo Frabetti
Diseño cubierta: Neslé Soulé
Jefe de Producción: Ricardo Prats

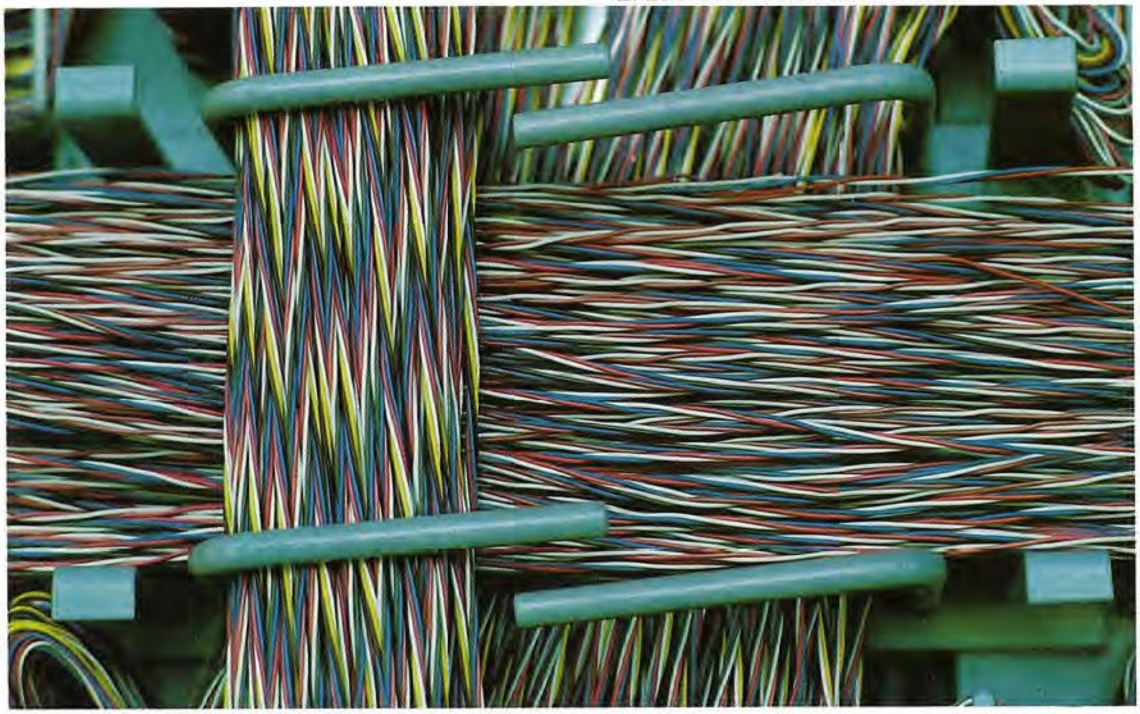
Composición electrónica: ITC-Fototipo. Barcelona-Madrid.
Imprime: CAYFOSA - Carretera de Caldas, Km. 3,7 - Santa Perpetua de Mogoda (Barcelona)
Depósito Legal: B. 37.099/83
ISBN (Obra completa): 84-7574-040-5
ISBN (Volumen IV): 84-7574-232-7
ISBN (Fascículos): 84-7574-044-8
Impreso en España - Printed in Spain
Distribuye: Marco Ibérica, Distribuidora de Ediciones, S.A.
Carretera de Irún, Km. 13,350, variante de Fuencarral, MADRID-34

El editor agradece la colaboración de:
Alfa Romeo, Buffetti Data, Commodore Italiana, CPT Italia, Creazioni Walt Disney, Data General, Digital, Doxa, Elsag, Ericsson, Facit Data Products, Ferrari, FIAT, Harden Italia, Hengstler Italia, Hewlett Packard, IBM, Intema, IRET Informática, Italcable, Lancia, Litton BEI, MEE, MSI Data Italia, Olivetti, Perkin-Elmer, Plessey Trading, Prime Italia, Rank Xerox, Rhône-Poulenc Italia, Sanco Ibex Italia, Sarin, Selca Elettronica, Selenia, Sperry, SIP, Telespazio.

El lenguaje Assembler

En cualquier ordenador personal, micro o mini, hay el microprocesador. Llamado también CPU, es decir, Central Processing Unit (unidad central de proceso), el microprocesador es un conjunto de circuitos, transistores, diodos, resistencias y condensadores que ocupan un espacio muy reducido. Piénsese que es posible contener hasta 450.000 transistores en un cuadrado de 5,5 mm de lado: en una cabeza de aguja podrían caber unos 25.000 transistores.
Las funciones realizadas por el microprocesador son principalmente dos: el control de los componentes que constituyen el calculador y la ejecución de los cálculos aritméticos y lógicos. Estas tareas se realizan bajo el control de un programa escrito en un lenguaje muy diferente al Basic: el lenguaje máquina, el único lenguaje que el microprocesador puede entender. En un lenguaje máquina, las instrucciones están constituidas por secuencias de estados lógicos 0 y 1, organizados en bytes o en palabras, según el tipo de microprocesador utilizado.
Cada instrucción realiza una misión muy definida y limitada, por lo que es necesaria una larga secuencia de instrucciones incluso para efectuar una sencilla operación.
La escritura de un programa directamente en lenguaje máquina no es una cosa sencilla. Se corre el riesgo de cometer muchos errores, y a menudo se obtienen programas que es prácticamente imposible comprender o modificar.
En la programación de un ordenador personal en un lenguaje de alto nivel como el Basic, la misión de traducir las instrucciones y los comandos en las correspondientes instrucciones en lenguaje máquina la realiza un conjunto de programas residentes establemente en memoria (RAM o ROM), como el intérprete, el compilador y el revisor. Sin embargo, en algunos casos, para que el ordenador personal realice determinadas misiones no es posible utilizar sólo el lenguaje de alto nivel.
Si por una parte los lenguajes de alto nivel interpretados son muy flexibles (como se dice, están orientados hacia el hombre), por otra tienen la desventaja de ser muy lentos en la ejecución. Efectivamente, en esta fase debe reclamarse continuamente el programa intérprete, que traduce cada instrucción Basic sencilla en la correspondiente serie de instrucciones en len-

El bus de conexión externa de un ordenador.



A. Tiracchia/AFE

guaje máquina, directamente ejecutables por el microprocesador. Esta lentitud es totalmente inaceptable en determinadas aplicaciones, como por ejemplo los gráficos por ordenador y los videojuegos, en los que se tiene la necesidad de producir animaciones en la pantalla, que necesitan rápidos desplazamientos en memoria de grandes cantidades de datos.
Por tanto, surge la necesidad de programar el ordenador en lenguaje máquina.
Para conseguir este objetivo de manera más cómoda y ágil cuando no se puede usando directamente los códigos hexadecimales, se utiliza el lenguaje Assembler que, a pesar de ofrecer la característica de ser muy similar al lenguaje máquina, evita la necesidad de emplear los códigos numéricos. A cada instrucción Assembler corresponde una instrucción en lenguaje máquina o bien una secuencia de estados lógicos 0 y 1 contenidos en general en uno o más bytes.
Respecto al lenguaje máquina, el Assembler es más manejable, y pone a disposición del programador algunos instrumentos necesarios para una ágil escritura de los programas, como la posibilidad de definir etiquetas (label) o bien nombres simbólicos para las variables.
La necesaria traducción de un programa del Assembler al lenguaje máquina se obtiene utilizando después un programa de sistema llamado **Ensamblador**.
Antes de analizar las características de este nuevo lenguaje y el funcionamiento del ensamblador daremos una descripción general de un microprocesador tipo. Esta fase de la exposición es prácticamente obligada porque, al ser el Assembler un lenguaje similar al lenguaje máquina, está muy ligado al funcionamiento del microprocesador, cuyas características son en buena parte establecidas por la estructura del microprocesador utilizado.

Estructura y funcionamiento del microprocesador

Un microprocesador puede tener una lógica de funcionamiento particularmente suya, pero existen algunas necesidades fundamentales a respetar, es decir, funciones que normalmente deben estar presentes.
Un microprocesador tiene en su interior un conjunto (set) de instrucciones codificadas que permiten la ejecución de operaciones bien definidas. Una secuencia de estas instrucciones residentes en la memoria está definida por códigos de programa.
En la memoria hay presentes tanto las funciones como los datos necesarios para la ejecución de las mismas. Generalmente, en las arquitecturas más sencillas, los datos sobre los cuales la instrucción actúa están escritos en posiciones de memoria contiguas a las ocupadas por la propia instrucción. En las arquitecturas más complejas es posible dividir la memoria en dos zonas: el **área de códigos**, en la que hay contenidas las instrucciones para la máquina, y el **área de datos**, que contiene los datos necesarios para realizar las instrucciones. En estos casos, el conjunto de las instrucciones del programa puede cargarse en una memoria de sólo lectura (ROM) y los datos, diferentes en cada ejecución, en una memoria de lectura y escritura (RAM) (ver a este respecto el gráfico de la página siguiente).
Por ejemplo, si se quiere calcular la suma de dos números y memorizar el resultado, el programa Basic debería estar constituido así (el código LET puede no estar):

```
10 LET V = 10
20 LET S = V + 20
```

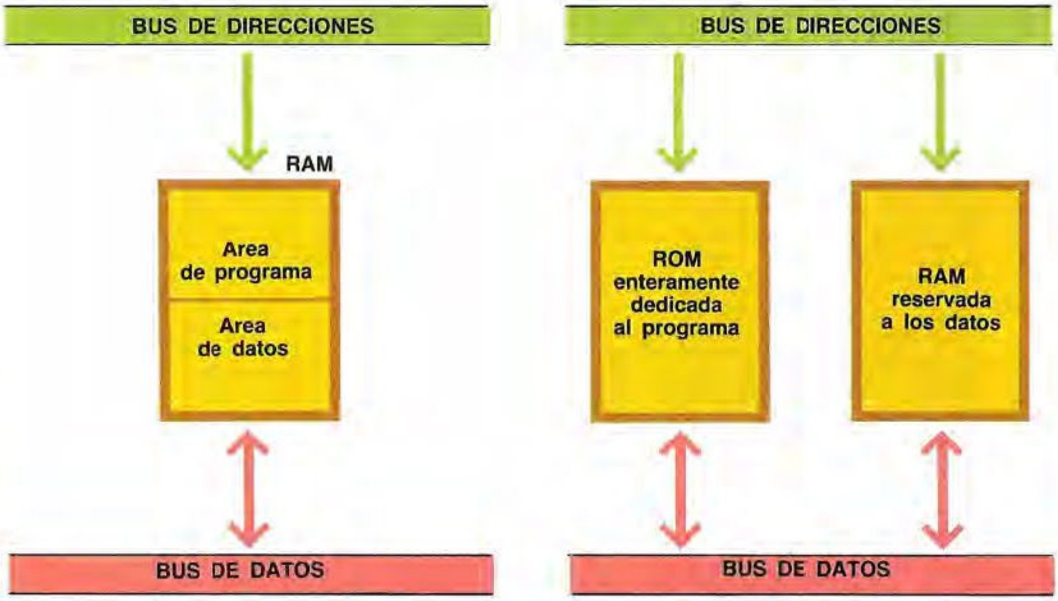
Las operaciones que la CPU debe realizar son las siguientes:

1 / Tomar y memorizar en una memoria interna el valor 10
2 / Tomar el valor 20 y sumarlo al contenido de la memoria interna
3 / Depositar el valor resultante en la posición de memoria que tiene el nombre simbólico S

En el ejemplo, el valor numérico 10 se ha memorizado previamente en una posición de nombre V, mientras que el valor 20 se proporciona como dato numérico asociado directamente a la instrucción de suma (línea 20). Para tomar el valor 10, la CPU debe dirigirse a la memoria V, mientras que para calcular la suma debe utilizar un valor numérico (20) que sigue al código de la instrucción. La serie efectiva de las operaciones a realizar entonces deberá estar organizada de la siguiente manera:

1 / Toma el contenido de la memoria V
2 / Suma a este valor el número 20
3 / Deposita el resultado en la memoria S

MEMORIAS DE UN SISTEMA DE MICROCALCULADOR



En la pág. 868 se ha representado un ejemplo de cómo podría estructurarse y realizarse este programa en un microprocesador de 8 bits.

Como ya se vio al hablar de los sistemas con microcalculador (págs. 125 y siguientes), un microprocesador comprende en su interior diversos dispositivos que le permiten realizar un determinado conjunto de instrucciones según una precisa secuencia de pasos.

Uno de los elementos fundamentales del ordenador es el **reloj** (clock). Este dispositivo, que según el constructor puede o no estar integrado en el microprocesador, tiene una gran importancia, puesto que tiene la misión de sincronizar todas las operaciones que debe realizar la CPU activándola según una secuencia ordenada y predeterminada.

El término reloj indica también la señal eléctrica emitida por el oscilador, que asume un valor lógico 1 (ON) para un cierto intervalo de tiempo, cuya duración va desde unos 125 nanosegundos (1 nanosegundo = 10^{-9} segundos) hasta unos 55 nanosegundos o menos en los sistemas más rápidos, y en el intervalo siguiente, de igual duración, tiene un valor lógico 0 (ver gráfico en la parte superior de la pág. 869).

Esta señal eléctrica llega a la CPU permitiéndole iniciar una operación cualquiera en el instante en que dicha señal cambia de nivel, pasando del valor lógico 1 al valor lógico 0 (o viceversa). Cada operación realizada por la CPU puede durar uno o más intervalos de reloj, según el tipo de operación y el tipo de microprocesador.

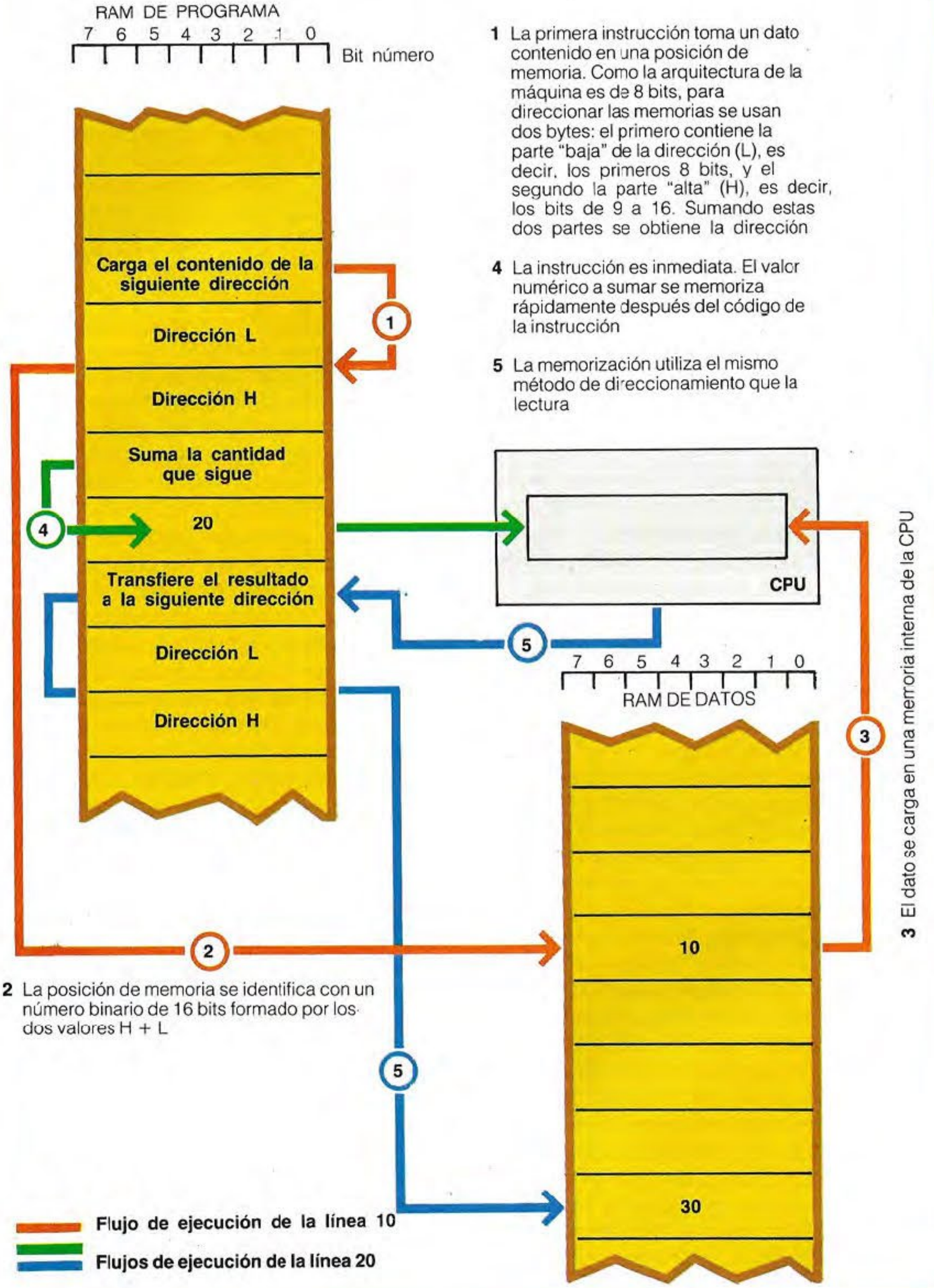
Ahora examinaremos con más detalle las fases que componen el funcionamiento operativo de la CPU. Se llama **ciclo de instrucción** el tiempo que necesita la CPU para leer en la memoria una instrucción cualquiera y realizarla. Una instrucción puede estar contenida en uno o más bytes de memoria. Se llama **ciclo de máquina** el intervalo de tiempo (para la máquina un intervalo de tiempo es un conjunto de estados lógicos del microprocesador) durante el cual la CPU descarga la dirección de un byte sobre el bus de direcciones y la memoria busca el elemento deseado y pone su contenido en el bus de datos (ver segundo gráfico de la pág. 869).

En realidad, el proceso descrito refleja sólo superficialmente el funcionamiento real de la CPU, que depende de forma sustancial del tipo de microprocesador.

Uno de los pasos fundamentales de un ciclo de instrucción es la codificación y la sucesiva ejecución de la instrucción.

Para descodificar lo que le ha transmitido la memoria, el microprocesador debe compararlo con el contenido de una tabla que enumera todas las instrucciones que se le indican.

EJECUCION DE UN PROGRAMA



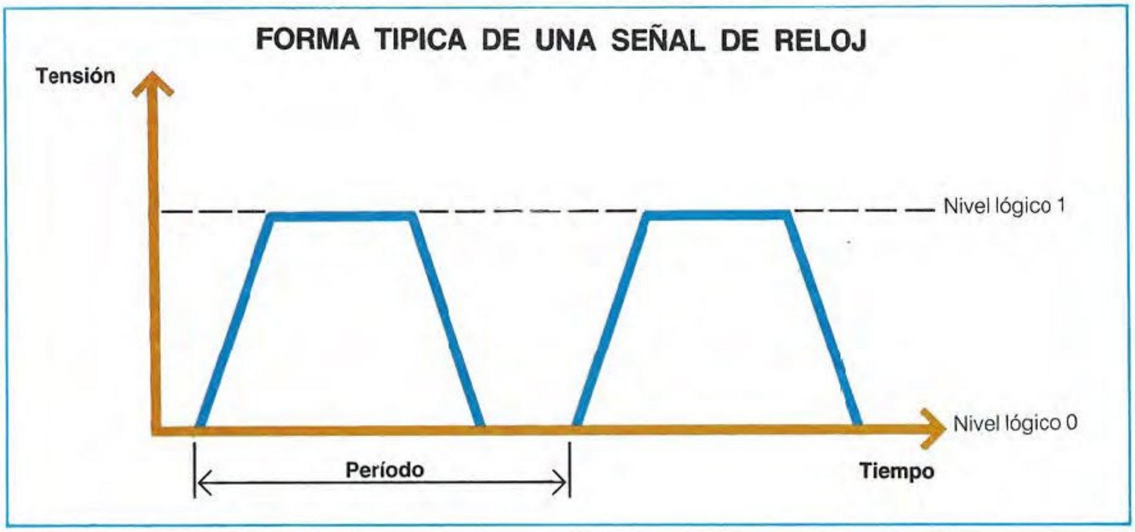
1 La primera instrucción toma un dato contenido en una posición de memoria. Como la arquitectura de la máquina es de 8 bits, para direccionar las memorias se usan dos bytes: el primero contiene la parte "baja" de la dirección (L), es decir, los primeros 8 bits, y el segundo la parte "alta" (H), es decir, los bits de 9 a 16. Sumando estas dos partes se obtiene la dirección

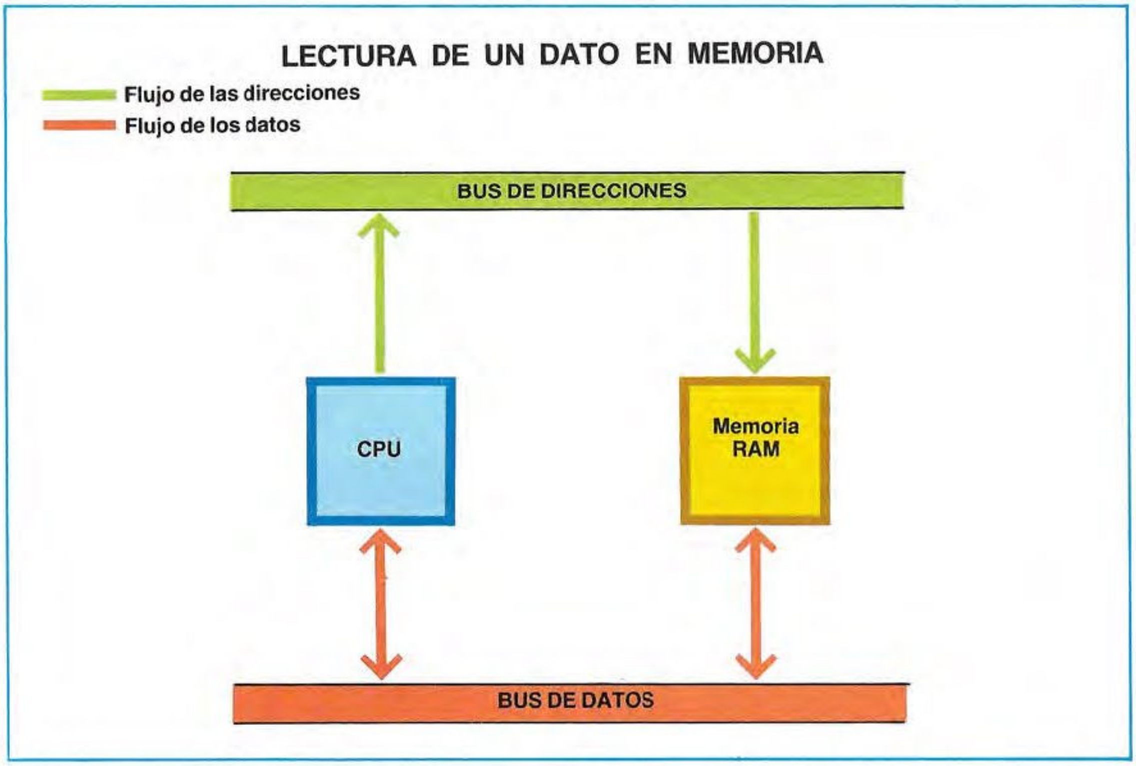
4 La instrucción es inmediata. El valor numérico a sumar se memoriza rápidamente después del código de la instrucción

5 La memorización utiliza el mismo método de direccionamiento que la lectura

2 La posición de memoria se identifica con un número binario de 16 bits formado por los dos valores H + L

3 El dato se carga en una memoria interna de la CPU





Cada microprocesador tiene un dispositivo llamado **unidad de control** (Control Unit) que, entre otras cosas, gestiona una memoria de sólo lectura (Control Store) que contiene los códigos de las instrucciones que el microprocesador puede realizar (ver gráfico de la pág. 870).

En el momento de la recepción de la instrucción, la CPU inicia uno o más ciclos de máquina según el número de bytes que acompañan dicha instrucción.

En general, lo que es necesario comprender de modo claro es el funcionamiento básico de la CPU, cuya descripción reemprenderemos cuando se traten más adelante los sistemas de microcalculador.

Después ilustraremos las características de algunos tipos de microprocesador en el comercio, para llegar a describir su lenguaje Assembler y los instrumentos disponibles para programar utilizando este lenguaje.

Los registros

Para realizar un programa, la CPU debe tener en su interior memorias de trabajo que le permitan realizar las operaciones deseadas depositando las informaciones necesarias y efectuando lecturas y modificaciones. Estas áreas se llaman **registros**.

Uno de los registros, utilizado para depositar los valores sobre los cuales deben realizarse los cálculos, recibe la denominación de **acumulador** (accumulator).

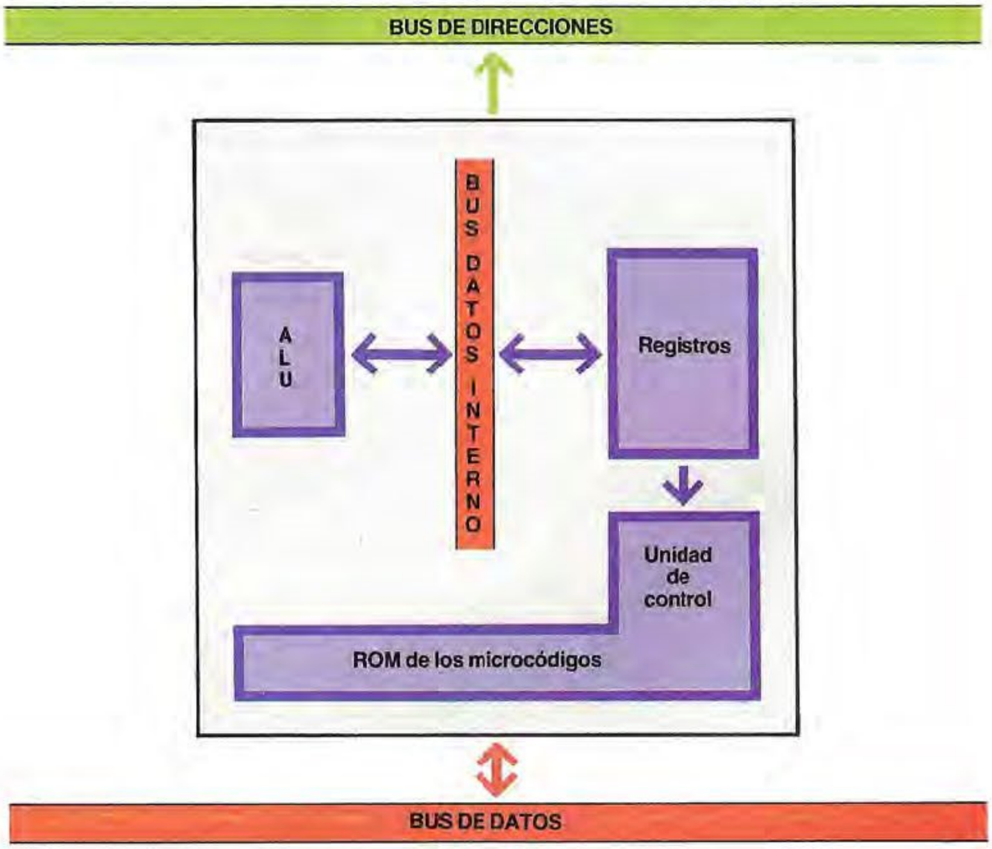
La presencia del acumulador permite que los cálculos puedan realizarse mucho más rápidamente (en el interior de la CPU) siempre que no sea posible obtenerlos operando directamente en la memoria de datos. Los valores existentes en la memoria de datos se modifican sólo al final del procedimiento de cálculo, que se realiza enteramente en la CPU.

En el ejemplo considerado en la pág. 868, el valor 10 se carga en el acumulador de la CPU y en el cálculo (siguiente instrucción), el valor 20 se suma a su contenido. El valor final se deposita en la memoria de datos, donde permanecerá hasta que no sea cancelado, bien porque se introduce un nuevo valor o bien porque se apaga la máquina.

La alternativa a este sistema es la de escribir en la memoria de datos el valor 20 para añadirlo (siempre en la memoria) al anterior.

Sin embargo, de este modo se tendría la necesidad de acceder a la memoria para cada dato a sumar. Si en lugar de sumar sólo dos números debiesen sumarse, por ejemplo, 7, la operación necesitaría 7 accesos: debería tomarse el primer número, introducirlo en la memoria, tomar el segundo número, sumarlo al primero y memorizar el resultado, tomar el tercer número, sumarlo al resultado anterior y así sucesivamente. Para disminuir el número de accesos a memoria, y por tanto para aumentar la velocidad de cálculo, se utiliza el acumulador como memoria de

COMPONENTES PRINCIPALES DE UNA CPU



tránsito durante todos los cálculos. En realidad, algunas numeraciones elementales se realizan directamente en la memoria de datos; por ejemplo, para incrementar en 1 el contenido de una memoria, existe una instrucción que no necesita la transferencia al acumulador, sino que realiza el incremento directamente. Otro registro indispensable es el **contador de programa**, en el que se memoriza, cada vez, la dirección de la instrucción que debe realizar el microprocesador. Este registro, llamado PC (Program Counter), contendrá un valor inicial que será la dirección de la localización de memoria que contiene la primera instrucción del programa a realizar. El ciclo de instrucción empieza cuando la CPU pone el contenido del contador del programa sobre el bus de direcciones y, por tanto, toma la primera instrucción de la memoria. Después, la CPU incrementa el contenido del contador del programa, por lo que el siguiente ciclo de instrucción irá a tomar la siguiente instrucción de la memoria. Si la instrucción ocupa más de una posición de memoria se toma en tiempos sucesivos, y la CPU incrementa cada vez el PC. Por ejemplo, la primera instrucción considerada en el gráfico de la pág. 868 (Carga el contenido...) está contenida en tres posiciones y, por tanto, debe tomarse en tres fases (la instrucción completa necesita cuatro ciclos de máquina: tres para la carga de los códigos y uno para la toma del dato deseado).

La CPU ejecuta las instrucciones de manera secuencial, a menos que una instrucción de salto (Jump o Branch) cambie el valor contenido en el contador de programa.

Otro registro generalmente presente en la CPU es el **registro de instrucción** (IR, Instruction Register) que contiene el código de la instrucción en tanto que ésta no puede descodificarse. Algunos de los actuales microordenadores tienen dos registros IR, con los que es posible tomar una instrucción mientras la anterior está en fase de ejecución. Esta técnica, llamada «pipelining», permite alcanzar velocidades de ejecución considerablemente elevadas.

En la CPU existen además algunos registros que sirven para encontrar las posiciones en que se encuentran los datos, posiciones que no son necesariamente secuenciales.

Para acceder a algún dato que reside en memoria (para una operación de lectura o de escritura) es necesario conocer la dirección de la posición donde reside el dato. Por tanto, siempre por motivos de velocidad, se crean los **contadores de datos** (data counters) o **registros de direcciones de memoria** (memory address registers) que contienen la dirección en memoria del dato examinado.

Funcionamiento de la unidad central de proceso

Para que la máquina pueda interpretar cada instrucción debe traducirse al correspondiente código en notación binaria, octal o hexadecimal.

El sencillo programa considerado anteriormente (V = 10, S = V + 20), en la simbología comprensible por el calculador se transforma en una larga serie de símbolos 1 y 0.

Para programar de este modo debe conocerse el código de cada instrucción a realizar.

Por ejemplo, supongamos que en nuestro programa las instrucciones se hayan representado de esta manera:

1 / Carga en el acumulador el valor contenido en la celda cuya dirección (simbólica) es V.
Código de instrucción = 0 0 1 1 1 0 1 0 = $= (2^5 + 2^4 + 2^3 + 2^1)_{10} = (58)_{10} = (72)_8 = (3A)_{16}$*.
La estructura que especifica la dirección (V) se explicará más adelante.

2 / Suma al acumulador el valor 20 (inmediato).
Código de instrucción = 1 1 0 0 0 1 1 0 = $= (2^7 + 2^6 + 2^2 + 2^1)_{10} = (198)_{10} = (306)_8 = (C6)_{16}$.

3 / Memoriza el valor contenido en el acumulador en la celda cuya dirección es S.
Código de instrucción = 0 0 1 1 0 0 1 0 = $= (2^5 + 2^4 + 2^1) = (50)_{10} = (62)_8 = (32)_{16}$.

En este punto debe proporcionarse, para cada una de las instrucciones que se utilizan, la dirección en que tomar o depositar los datos.

Supongamos que el programa empieza en la posición de memoria $(100)_8$ o $(40)_{16}$, y que los datos están memorizados a partir de la posición $(300)_8$ o $(C0)_{16}$. Los datos (valores decimales 10 y 20) en representación binaria, octal y hexadecimal son

$(10)_{10} = 00001010 = (12)_8 = (A)_{16}$
$(20)_{10} = 00010100 = (24)_8 = (14)_{16}$

* La notación $(58)_{10}$ indica el número 58 en base 10, así como las notaciones $(72)_8$ y $(3A)_{16}$ indican el mismo número expresado respectivamente en base octal y hexadecimal.

Estos valores deberán memorizarse en las oportunas posiciones de memoria para poder ser utilizados después por el programa.

El aspecto final completo del programa se representa en esta página.

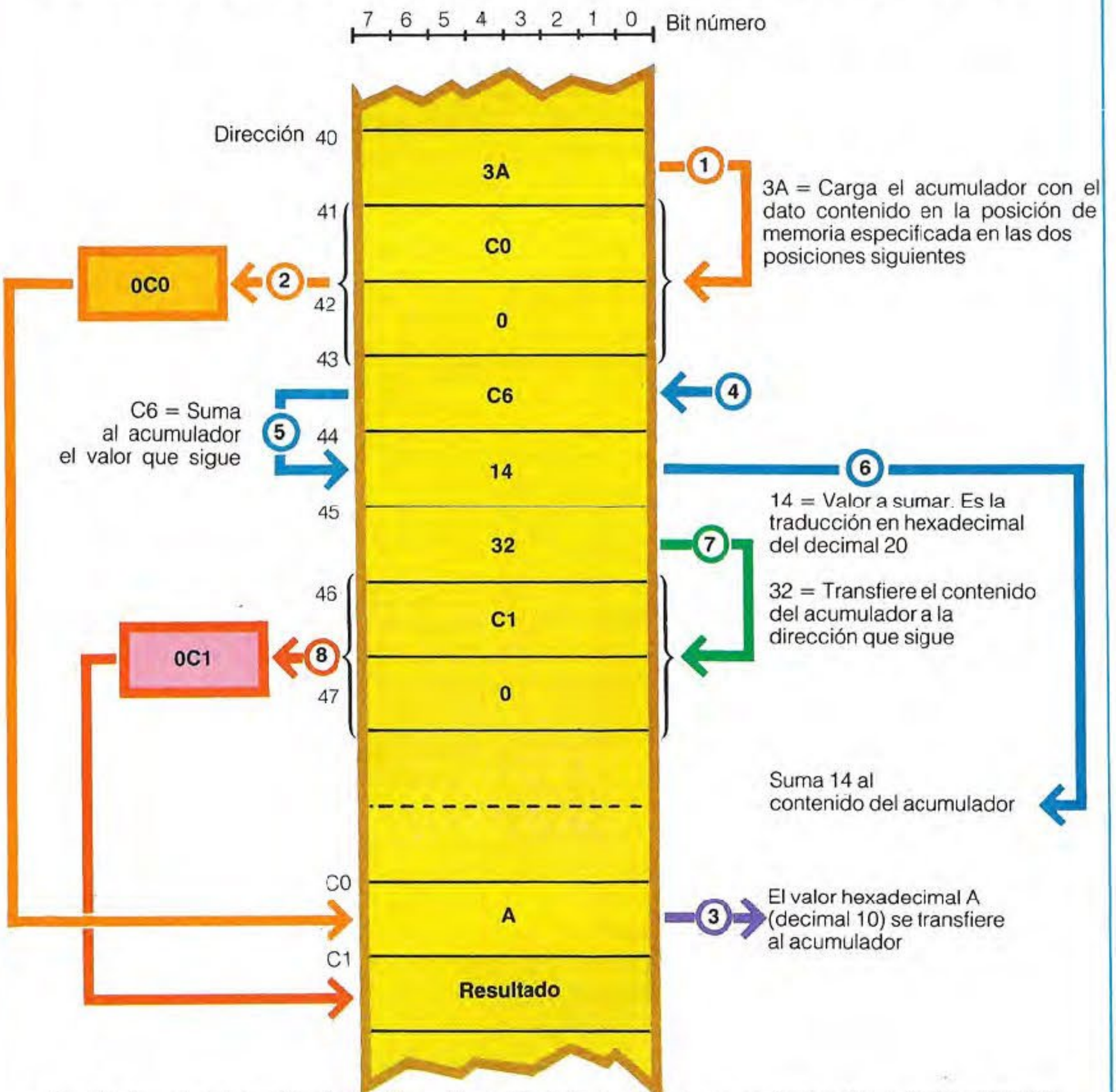
En cambio, en las págs. 873 a 875, la ejecución del programa se realiza paso a paso. La lectura de las indicaciones proporcionará una orientación para comprender cuál es el desarrollo del proceso de gestión en un microprocesador.

Para simplificar posteriormente la descripción, en el ejemplo se ha supuesto que pueden indicarse las direcciones con sólo 8 bits.

El empleo de los códigos hexadecimales para escribir un programa aun extremadamente sencillo, presenta dos principales dificultades:

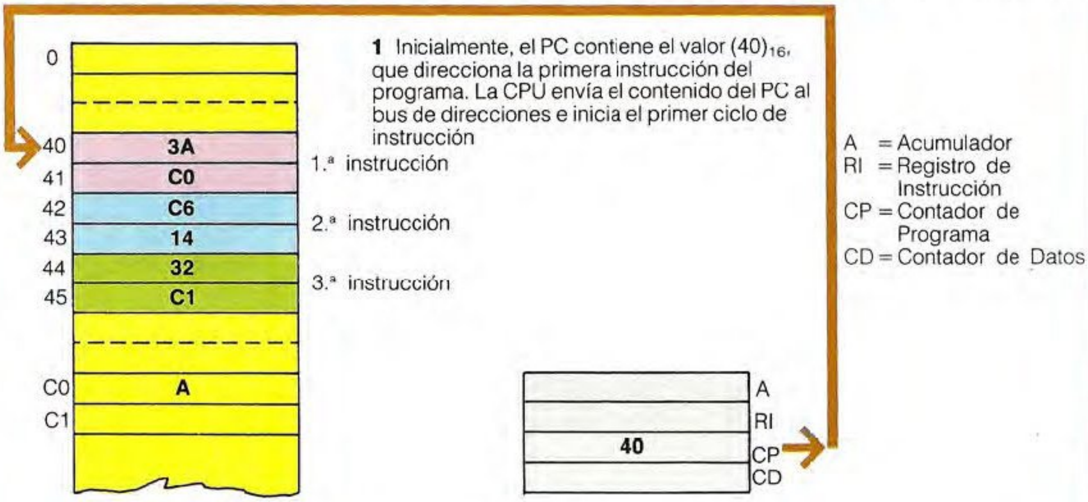
— las instrucciones (expresadas como códigos hexadecimales) no pueden identificarse fácilmente. La verificación del programa se convierte en una empresa muy ardua, que suele conducir a resultados inciertos

EJECUCION DE UN PROGRAMA ESCRITO EN LENGUAJE MAQUINA

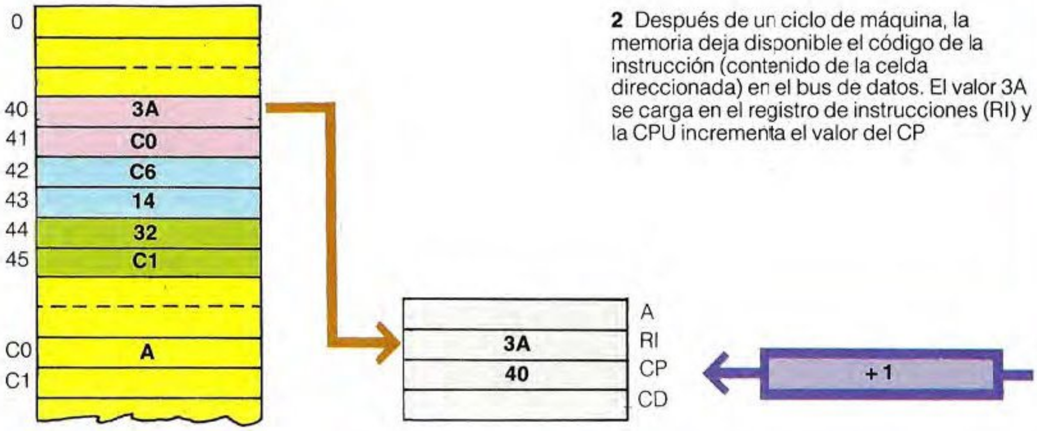


En este ejemplo se han utilizado dos tipos diferentes de instrucción de carga. En la primera, la del código 3A (Carga el acumulador), los datos se toman a través de un direccionamiento a una posición cualquiera de memoria (en el ejemplo C0), también distante de la que contiene la instrucción. El segundo tipo (inmediato, código C6) utiliza como dato (en el ejemplo 14) el contenido de la memoria que sigue a la que contiene el código de instrucción

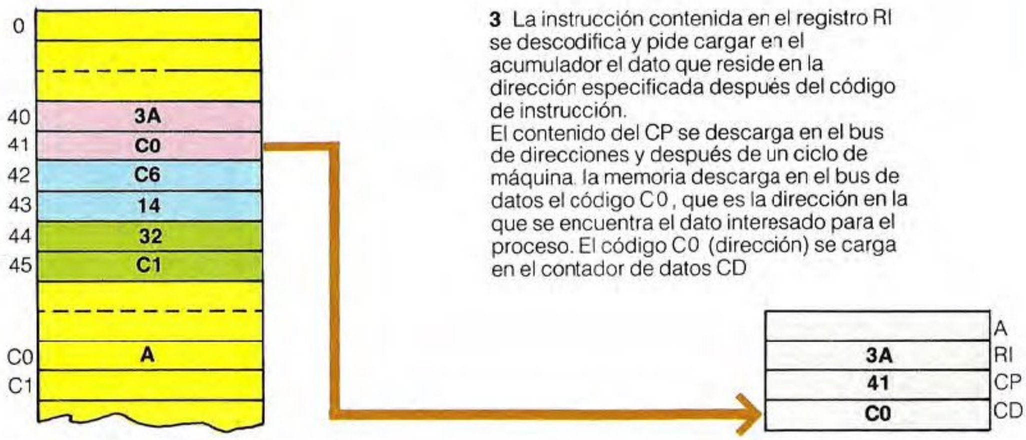
VARIACIONES DEL CONTENIDO DE LOS REGISTROS EN FASE DE EJECUCION



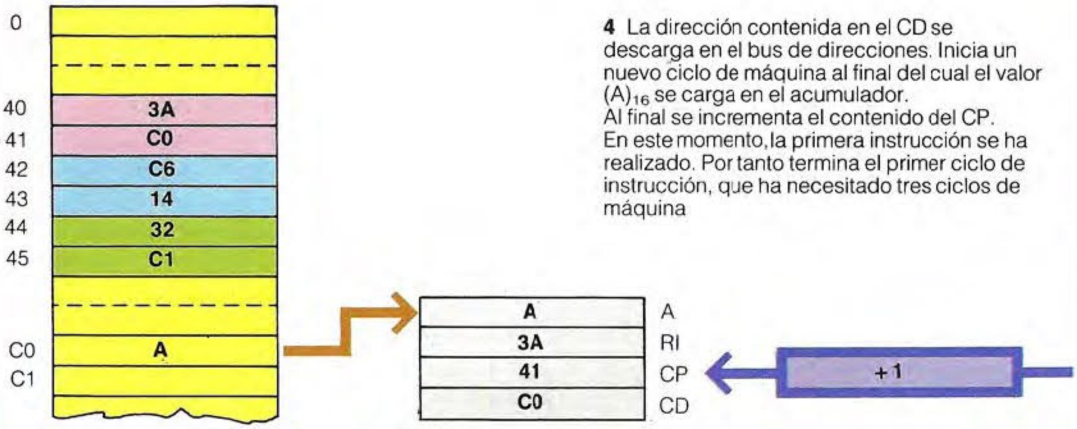
1 Inicialmente, el PC contiene el valor $(40)_{16}$, que direcciona la primera instrucción del programa. La CPU envía el contenido del PC al bus de direcciones e inicia el primer ciclo de instrucción



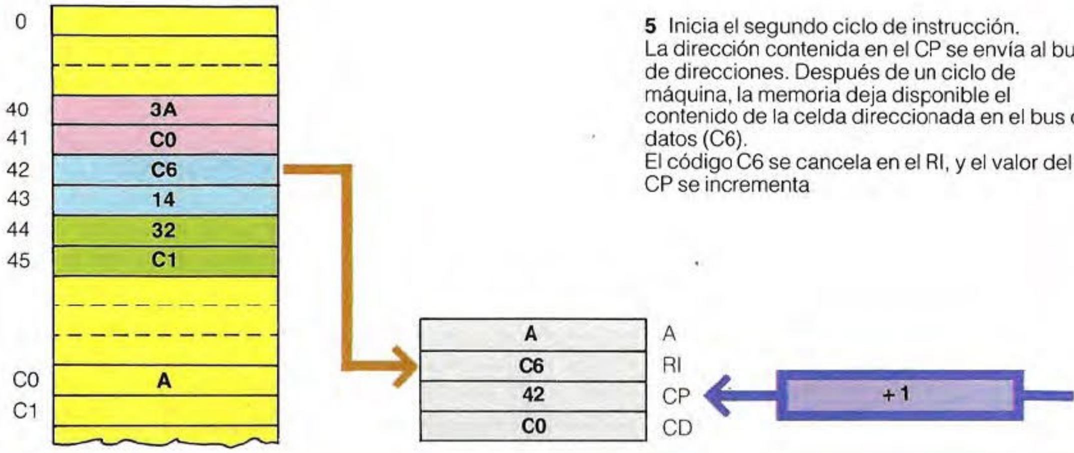
2 Después de un ciclo de máquina, la memoria deja disponible el código de la instrucción (contenido de la celda direccionada) en el bus de datos. El valor 3A se carga en el registro de instrucciones (RI) y la CPU incrementa el valor del CP



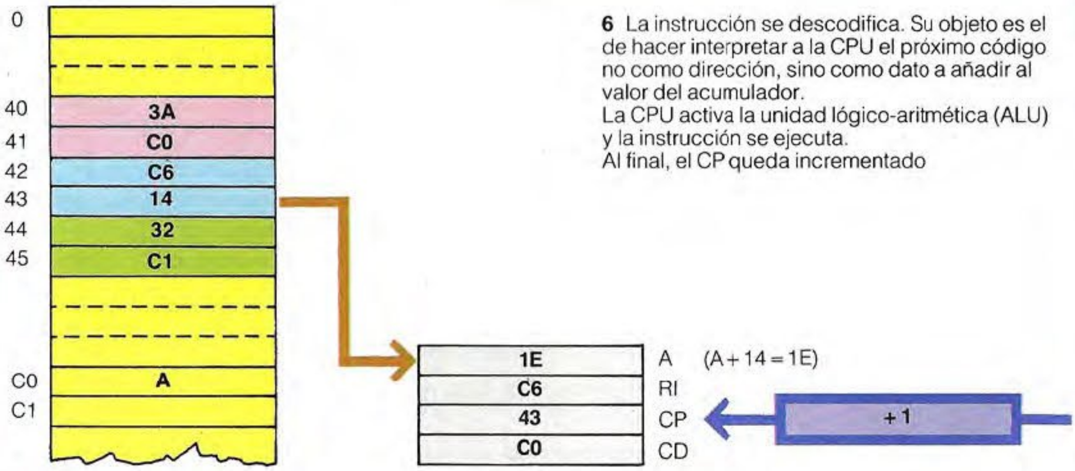
3 La instrucción contenida en el registro RI se descodifica y pide cargar en el acumulador el dato que reside en la dirección especificada después del código de instrucción.
El contenido del CP se descarga en el bus de direcciones y después de un ciclo de máquina, la memoria descarga en el bus de datos el código C0, que es la dirección en la que se encuentra el dato interesado para el proceso. El código C0 (dirección) se carga en el contador de datos CD



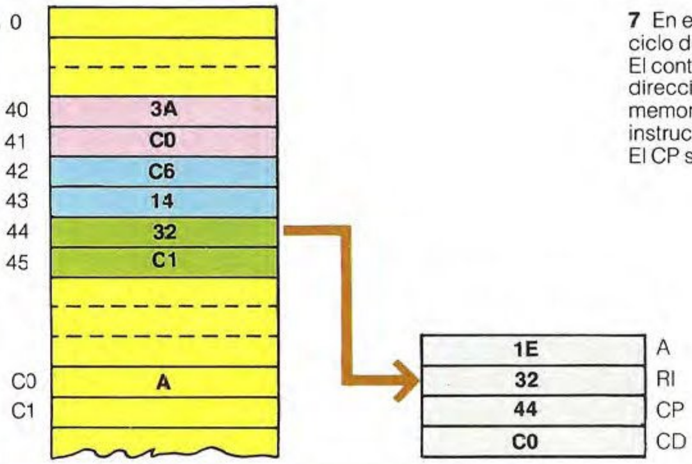
4 La dirección contenida en el CD se descarga en el bus de direcciones. Inicia un nuevo ciclo de máquina al final del cual el valor $(A)_{16}$ se carga en el acumulador.
Al final se incrementa el contenido del CP.
En este momento, la primera instrucción se ha realizado. Por tanto termina el primer ciclo de instrucción, que ha necesitado tres ciclos de máquina



5 Inicia el segundo ciclo de instrucción.
La dirección contenida en el CP se envía al bus de direcciones. Después de un ciclo de máquina, la memoria deja disponible el contenido de la celda direccionada en el bus de datos (C6).
El código C6 se cancela en el RI, y el valor del CP se incrementa

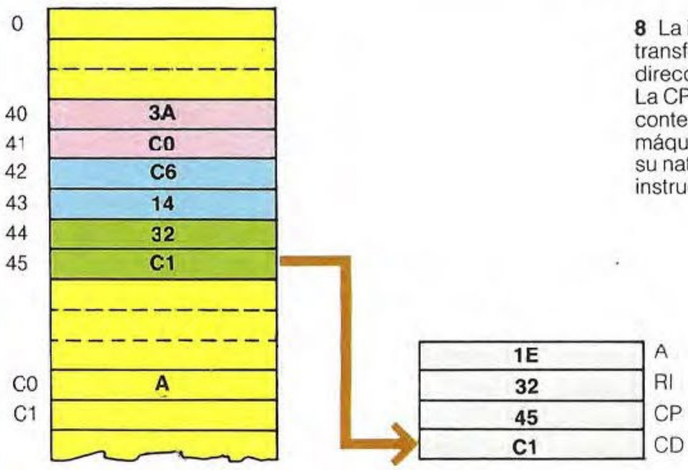


6 La instrucción se descodifica. Su objeto es el de hacer interpretar a la CPU el próximo código no como dirección, sino como dato a añadir al valor del acumulador.
La CPU activa la unidad lógico-aritmética (ALU) y la instrucción se ejecuta.
Al final, el CP queda incrementado

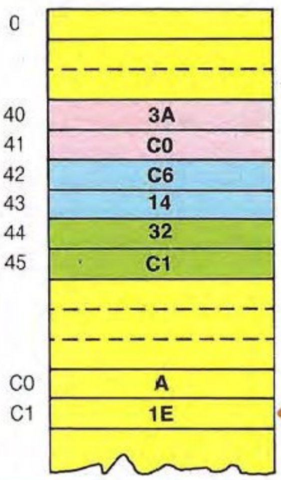




7 En este punto se termina también el segundo ciclo de instrucción y se inicia el tercero.
El contenido del CP se descarga en el bus de direcciones. Después de un ciclo de máquina, la memoria deja disponible el código de instrucción que se carga en el RI.
El CP se incrementa



8 La instrucción se descodifica: el sistema debe transferir el contenido desde el acumulador a la dirección que sigue al código de instrucción.
La CPU descarga en el bus de datos el contenido del CP. Después de un ciclo de máquina, el valor C1 se cargará en el CD, dada su naturaleza de dirección y el tipo de instrucción.





9 Tercer ciclo de máquina de la tercera instrucción.
La CPU toma el contenido del acumulador y lo deposita en la celda direccionada por el CD.
Esta operación concluye la ejecución del programa.

— el direccionamiento de las variables, que no utiliza nombres simbólicos, se limita a recordar al programador el contenido de cada posición de memoria

Para eliminar estos inconvenientes se ha creado el lenguaje Assembler que, a pesar de estar orientado más a la máquina que al usuario, permite una programación relativamente sencilla. Las instrucciones Assembler se identifican con un código mnemónico, compuesto normalmente de tres letras, y las direcciones pueden proporcionarse con una etiqueta (label). Por ejemplo, normalmente las instrucciones «carga el acumulador A con...», es decir, el código 3A, tienen como simbolismo las letras LDA (LoaD A); asociando a la memoria C0 la etiqueta XY, la instrucción completa se convierte en

LDA XY = carga A con el contenido de la
(3A) (C0) memoria de nombre XY

Arquitectura interna del microprocesador

Un microprocesador comprende en su interior un bus de comunicaciones entre los diversos componentes y un conjunto de registros.
Generalmente se dice que un microprocesador es de 8 bits si el acumulador (uno o más registros de este tipo) es de 8 bits, de 16 bits si es de 16 bits, etc. Se comprende que es conveniente que la amplitud del bus interno, utilizado para todas las comunicaciones entre los registros y la ALU y para el intercambio de informaciones con el exterior, esté estructurado con el mismo número de bits del acumulador.
En realidad no siempre es así. Por ejemplo, el Motorola 68000 está constituido por 16 registros de 32 bits, pero su bus de datos interno (y externo) es de 16 bits.
Esta limitación se debe solamente a los límites impuestos por la tecnología empleada, que no permite insertar un número de patillas superior a 64 en la cápsula del microprocesador.
Esta cápsula se llama DIP (Dual In-line Package) y es el soporte que permite conectar el chip a los otros circuitos del sistema.
Otro parámetro que caracteriza a los microprocesadores es el número de bits que viajan al mismo tiempo, en un determinado momento, por el bus de direcciones.
Este parámetro es muy importante como índice de las prestaciones del microprocesador, puesto que indica cuántas celdas de memoria pueden ser direccionadas directamente por la CPU, es decir, a cuántas celdas puede acceder con un único ciclo de máquina.
Si un bus de direcciones es de 16 bits, generalmente significa que existen 16 patillas en el microprocesador, cada una de las cuales está conectada a una pista diferente del circuito impreso.
Utilizando 16 bits es posible direccionar directamente $2^{16} - 1 = 65535$ celdas de memoria.
Dado que generalmente cada celda de memoria contiene 8 bits, significa que es posible direccionar 65535 celdas de un byte, o también la memoria visible puede ser de 64 kbytes como máximo (recuérdese que 1 kbyte vale 1024 bytes y que 1 Mbyte vale 1.048.576 bytes).
Principalmente para mantener bajo el costo del microprocesador, varios fabricantes tienen en catálogo productos que no presentan 16 patillas para las direcciones en el chip, pero que también pueden direccionar 65536 celdas de memoria y a veces también más. En estos casos, durante un ciclo de máquina se envía al bus de direcciones (formado por ejemplo por 8 pistas) la primera mitad de la dirección (normalmente la parte más alta de la dirección) y en el ciclo de máquina siguiente, la segunda mitad.
En este caso se mantiene el valor máximo de direccionamiento, pero para cada acceso a la memoria se emplea el doble de tiempo, y esto incide en las prestaciones de todo el sistema.
Por ejemplo, el microprocesador Intel 8088 tiene una arquitectura interna de 16 bits (tanto para los registros como para el bus interno) pero comunica con el exterior sólo con 8 bits.
Del mismo fabricante existe el modelo 8086 que interiormente es idéntico al 8088, pero dispone de 20 líneas para el direccionamiento.
También debe tenerse presente que tener pocos hilos simplifica notablemente las conexiones. Por otra parte, el costo de la memoria siempre es más bajo, mientras que aumentan las necesidades de tener siempre más espacio disponible para datos y programas que deben elaborarse velozmente.
Por este motivo, muchos microprocesadores de la nueva generación tienen arquitecturas que ya no son de 8, sino de 16 bits, y algunos ya de 32 bits, con 20 a 32 bits de direccionamiento. La ampliación de los bus y de los registros también permite mejorar la precisión de los cálculos. En un acumulador de 8 bits, el número más grande

que puede representarse es 256, mientras que si el acumulador es de 16 bits, se llega a 65636, y si es de 32 bits, a más de 4.000 millones. La memoria direccionable va desde el megabyte (20 bits de direccionamiento, Intel 8086) y 16 megabytes (24 bits de direccionamiento, Motorola 68000) a los 500 megabytes (29 bits de direccionamiento, Hewlett-Packard 9000).
A continuación supondremos que se trabaja con un microprocesador que dispone de una arquitectura de 8 bits (acumulador y bus interno) y que comunica exteriormente con 16 bits (Program Counter, Data Counter).

Resumiendo:
Los registros son celdas de memoria interna en la CPU. Los datos y las instrucciones pueden guardarse en un registro hasta que el bus u otro dispositivo no esté preparado para recibirlos. La CPU puede tomar datos de los mismos sin acceder a la memoria. Los registros y el bus interno de la CPU normalmente tienen la misma amplitud máxima (número de bits).

El costo de las conexiones internas limita el número de los registros en el interior de la CPU, pero la tendencia actual de tener microprocesadores siempre más rápidos está conduciendo a utilizar un elevado número de registros.

Unidad lógico-aritmética (ALU)

Además de los registros, la CPU contiene un conjunto de componentes necesario para realizar algunas funciones esenciales de cálculo: este conjunto toma el nombre de **ALU** (Arithmetic Logic Unit, unidad lógico-aritmética).
Los dispositivos que componen una ALU permiten que el microprocesador realice las siguientes operaciones (ver el gráfico en la parte superior de la página siguiente):

— sumas
— complemento de una palabra
— restas (no siempre)
— operaciones lógicas (AND, OR, NOT)
— desplazamiento (shift) de uno o más bits a la izquierda o a la derecha.

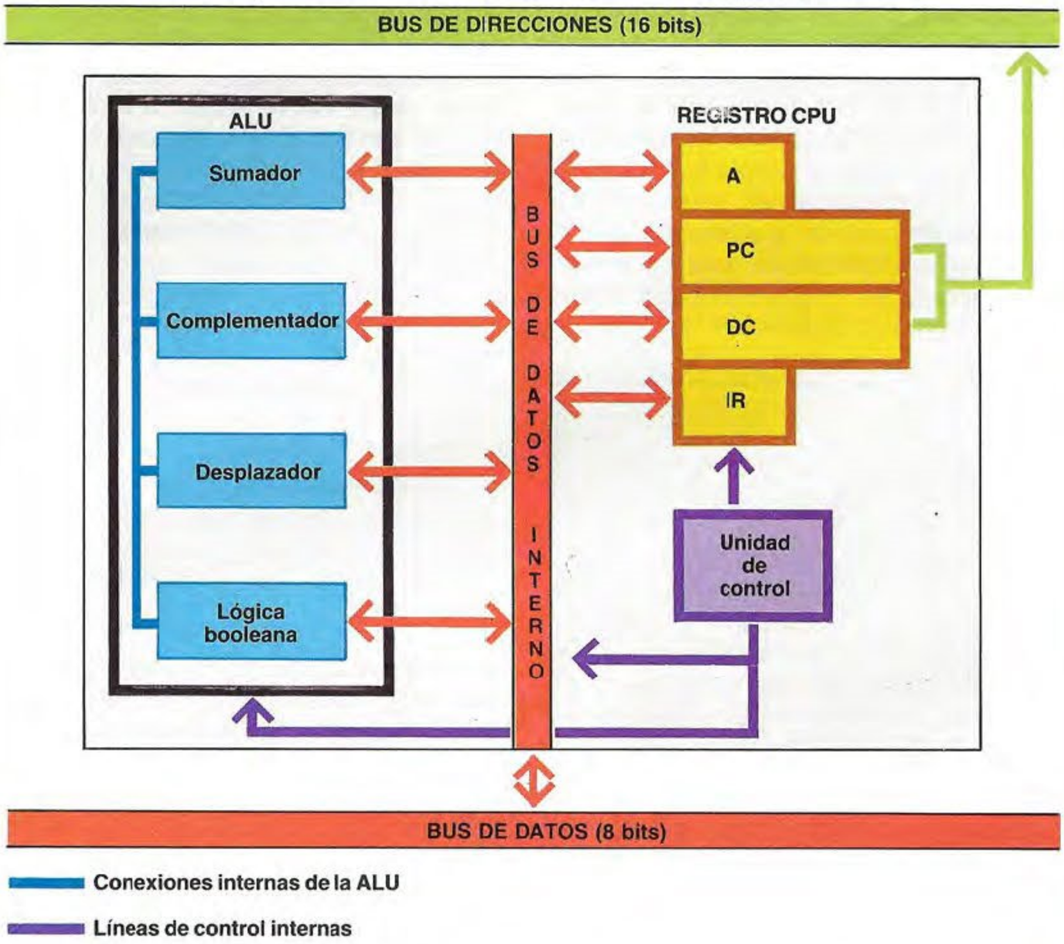
Evidentemente, todas esas operaciones pueden realizarse sólo en números binarios.
Para poder comprender el funcionamiento de una ALU, por tanto, será necesaria una descripción más particularizada de las reglas de la aritmética binaria. En consecuencia, a continuación se mostrarán los circuitos y dispositivos que realizan estas reglas.

Suma binaria. La suma binaria sigue reglas muy similares a las válidas en base decimal, con la ventaja de ser mucho más sencillas.
La operación fundamental es la suma de dos bits; las posibles combinaciones de dos bits a sumar son las siguientes:

Primer sumando (A)	Segundo sumando (B)	Suma (S)	Acarreo (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

El primer sumando se ha indicado con el símbolo A, el segundo con B, la suma de los dos bits con S y el acarreo con el símbolo C, de la palabra inglesa carry. En cada ALU siempre hay un circuito llamado **sumador** (adder) que puede efectuar la suma de dos números binarios de longitud igual a 8 o a 16 bits, según el tipo de microprocesador.
Antes de pasar a la descripción de un circuito sumador para números de 8 o 16 bits, analizaremos cómo puede realizarse un sumador para dos bits.
La función S (suma) puede realizarse con el circuito lógico indicado en la pág. 878 abajo, donde el acarreo C se ha realizado con un simple circuito AND, como puede comprobarse construyendo la tabla de verdad.
Un circuito como el mostrado se llama semisumador (half-adder).
Ahora podría pensarse en realizar un circuito sumador de 8 bits con 8 semisumadores puestos en paralelo como se muestra en la ilustración de la pág. 879, donde el circuito semisumador del ejemplo anterior se ha representado con bloques indicados con las letras HA.
Sin embargo, esta solución no es aplicable, puesto que con el semisumador no pueden tenerse en cuenta eventuales acarreos generados por las etapas de más a la derecha. Efectivamente, siempre refiriéndonos al gráfico de arri-

ESTRUCTURA INTERNA DE UN MICROPROCESADOR



ESQUEMA DE SUMADOR DE DOS BITS (HALF-ADDER)

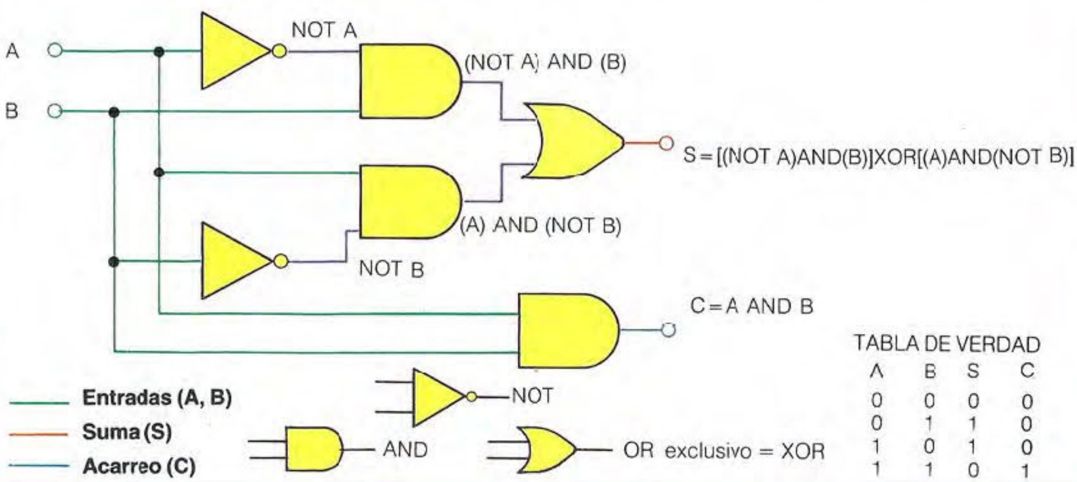


TABLA DE VERDAD

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

SUMADOR DE OCHO BITS

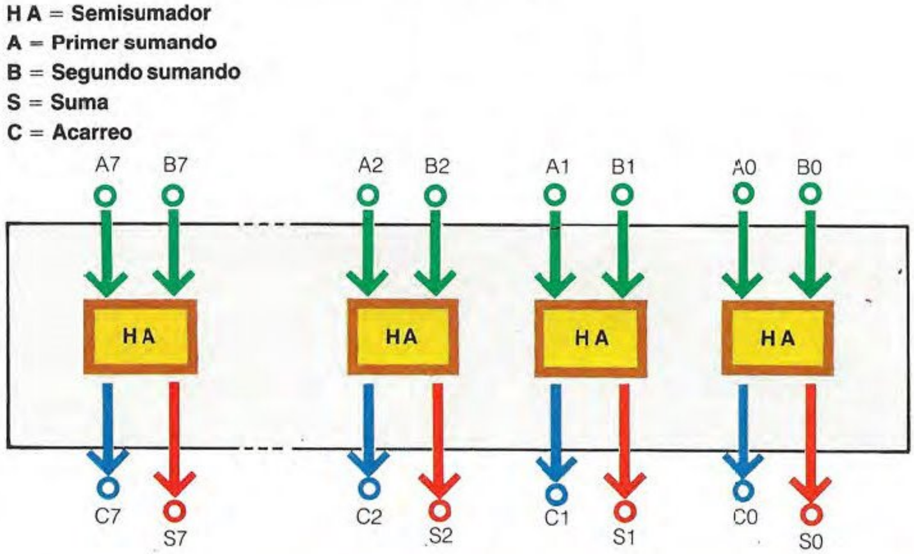


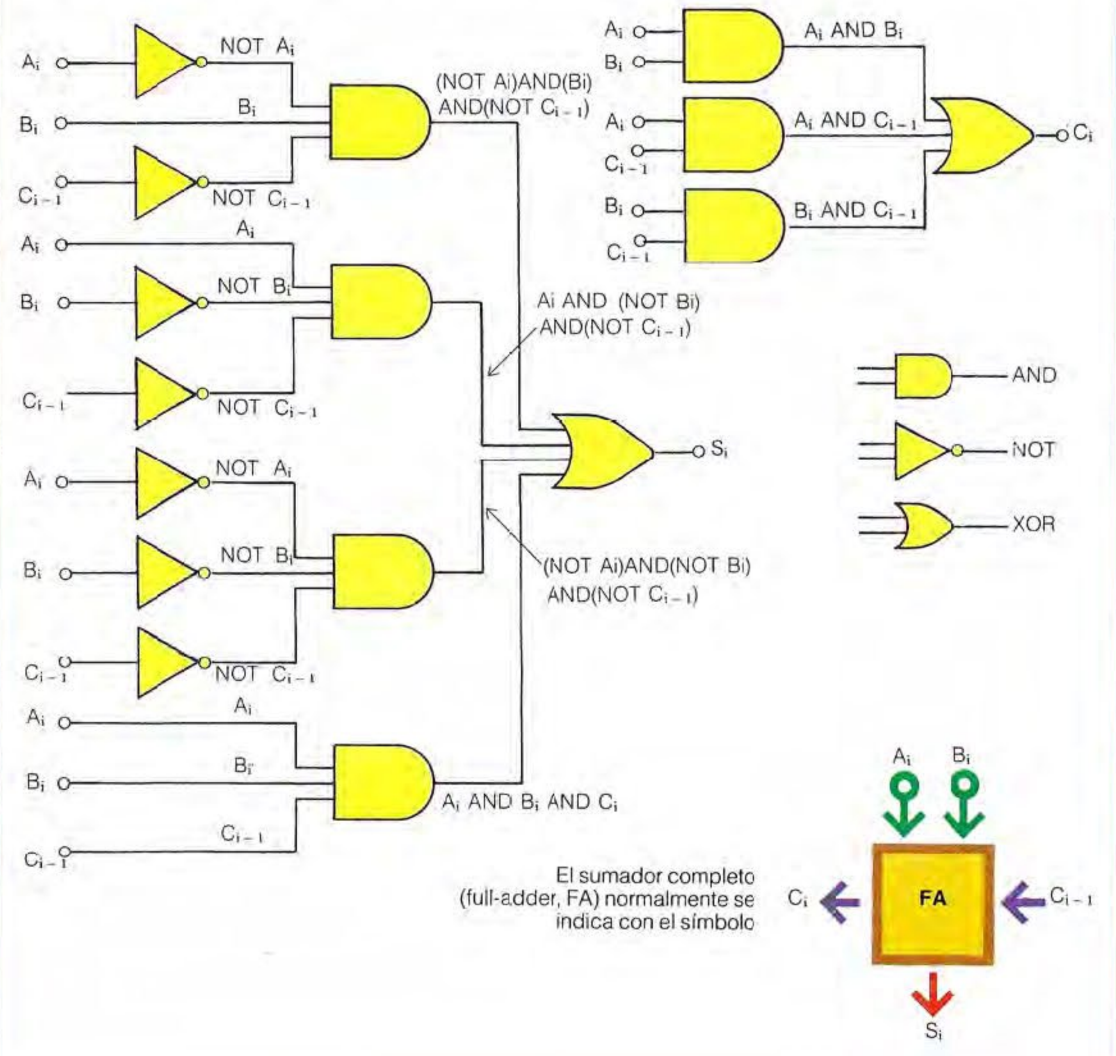
TABLA DE VERDAD DE UN SUMADOR COMPLETO (FULL-ADDER)

ENTRADAS			SALIDAS	
Acarreo anterior	Primer sumando	Segundo sumando	Suma	Acarreo
Ci-1	Ai	Bi	Si	Ci
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

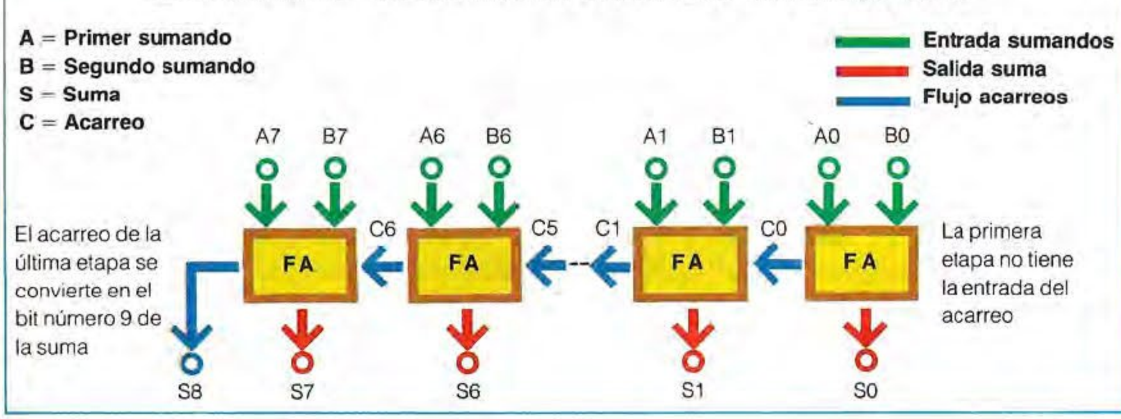
ba, si los bits menos significativos (A0 y B0) de los dos sumandos fuesen igual a 1, el acarreo C0 = 1 no podría sumarse a la siguiente etapa (junto con los bits A1 y B1) y, por tanto, no podría tenerse en cuenta.

Para superar este problema se recurre a la utilización del sumador completo, o full-adder, que puede sumar tanto los dos bits como el acarreo de la suma de los bits inmediatamente anteriores. La tabla de verdad de un sumador completo, mostrada aquí al lado, es más complicada que la del semisumador, así como su realización práctica (mostrada en la pág. 880, arriba). Una de las principales diferencias con respecto al circuito considerado anteriormente consiste en el tipo particular de AND utilizado, que debe prever tres entradas: dos para los bits de suma (como el circuito anterior) y uno para el acarreo de la etapa anterior. En el ejemplo citado, el sumador completo está dividido en dos partes: la primera realiza el cálculo de la suma (columna Si de la tabla de verdad), la segunda calcula el acarreo (columna Ci de la tabla de verdad).
Puede construirse un circuito que pueda sumar dos números de 8 bits teniendo en cuenta los acarreos y utilizando un cierto número de sumadores completos. Este circuito se ha esquematizado en la pág. 880, abajo. Está compuesto de tantos sumadores completos en paralelo como cuantos bits hay que sumar. Una vez generado el acarreo C0 de la suma de los dos bits menos significativos (A0 y B0), éste pasa a la siguiente etapa, donde se suma a los bits A1 y B1. Así se generan la suma S1 y el acarreo C1, que debe enviarse a la siguiente etapa.

ESQUEMA DE PRINCIPIO DE UN SUMADOR COMPLETO (FULL-ADDER)



ESQUEMA DE UN SUMADOR PARALELO DE OCHO BITS



El procedimiento se repite hasta la octava etapa en la que el acarreo C7 se convierte en el bit más significativo S8 de la suma.
Como puede verse, el resultado de la suma de 8 bits puede ser un número de 9 bits.

Resta binaria. El mismo procedimiento utilizado para la construcción de un sumador binario paralelo puede utilizarse para la construcción de un restador binario.
La tabla de verdad del circuito se muestra aquí abajo, donde D indica la diferencia entre los dos bits (A y B) y C el préstamo.
Cuando A = 0 y B = 1, para calcular la diferencia (0 - 1) es necesario tomar en préstamo un 1, de forma totalmente análoga a lo que se hace en la resta decimal.
Sin embargo, normalmente en las ALU de los microprocesadores no hay presente ningún circuito restador, puesto que la operación de sustracción puede hacerse fácilmente utilizando circuitos ya existentes en la ALU: el sumador es el complementador. Este hecho conduce evidentemente a una degradación de las prestaciones del sistema en términos de velocidad, pero hace la realización de la ALU más sencilla y menos costosa.

Complementación. La aritmética decimal permite ilustrar de forma sencilla el modo de realizar la resta con una operación de complementación y una suma. Restar un número decimal de otro equivale a sumar a este último el complemento a 10 del sustraendo.
El complemento a 10 de un número se obtiene restando de 10 el valor de dicho número. Por ejemplo, el complemento a 10 de 3 es 7 (10 – 3 = 7), porque para obtener el número 10 debe añadirse 7 a la cifra 3.
Para calcular una resta con el método del complemento, en primer lugar debe calcularse el complemento a 10 del sustraendo y después sumar el minuendo a este resultado, despreciando un eventual acarreo final.
Por ejemplo, considerando la resta en el sistema decimal:

$$8 - 3 = 5$$

el complemento a 10 de 3 es 7, por lo que se tiene

$$8 + 7 = 15$$

Por tanto, el resultado es 5 y debe despreciarse el acarreo 1.
Sin embargo, podría preguntarse por qué motivo se calcula la resta con la suma de un número complementado cuando, para encontrar el complemento del número también debe realizarse una resta.
El motivo es muy sencillo. El equivalente binario del complemento a 10 es el complemento a 2, que es muy fácil de calcular.
El complemento a 2 de un número binario se obtiene sustituyendo el símbolo 0 por el símbolo 1 y viceversa, y sumando 1 al número binario así obtenido.
La operación de sustitución del símbolo 0 por el 1 y viceversa se llama complemento a 1.
Por ejemplo, el complemento a 1 del número 1 0 1 0 1 1 1 0 es 0 1 0 1 0 0 0 1. El complemento a 2 de este último número será:

```
0 1 0 1 0 0 0 1 +    (complemento a 1)
              1 =
-----------------
0 1 0 1 0 0 1 0      (complemento a 2)
```

Ahora veremos cómo se realiza la resta binaria mediante la complementación. Consideremos a continuación la resta

```
    Minuendo          Sustraendo
1 1 1 0 1 0 1 0 – 1 0 1 0 1 1 1 0
```

El complemento a 2 del sustraendo ya se ha encontrado y es igual a 0 1 0 1 0 0 1 0, por lo que se tiene:

TABLA DE VERDAD DE UN RESTADOR BINARIO

Minuendo A	Sustraendo B	Diferencia D	Préstamo C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

```
  1 1 1 0 1 0 1 0 +
  0 1 0 1 0 0 1 0 =
  ─────────────────
┌→1 0 0 1 1 1 1 0 0     Resultado
└ Bit de acarreo
  a despreciar
```

Como en el caso decimal, el bit de acarreo se desprecia. Comprobemos la exactitud de la operación transformando los números binarios en decimales

	Binario	Decimal
Minuendo	1 1 1 0 1 0 1 0 −	234 −
Sustraendo	1 0 1 0 1 1 1 0 =	174 =
Resultado	0 0 1 1 1 1 0 0	60

En la práctica, en los circuitos de la ALU, la operación de complementación a 2 se realiza de forma diferente a lo que se ha descrito anteriormente.
Se toma el número a complementar, por ejemplo 1 0 1 0 1 1 1 0 y se lee de derecha a izquierda. Cuando se encuentra el primer 1 se deja inalterado, mientras que los siguientes símbolos se invierten, cambiando 0 por 1 y viceversa:

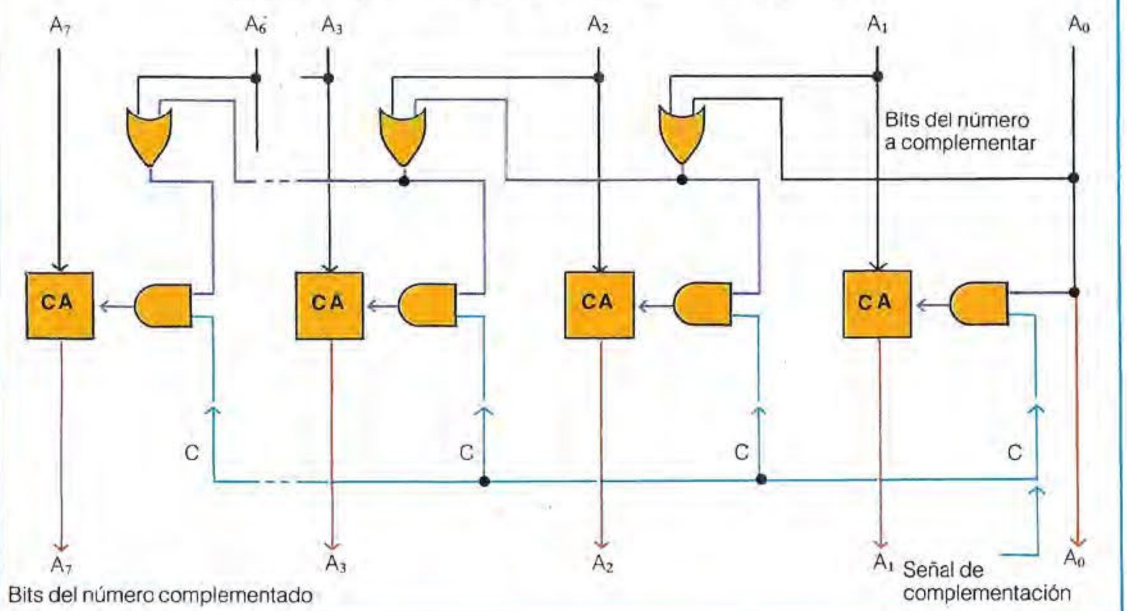
0 1 0 1 0 0 1 0

El resultado es idéntico al ya obtenido.

En esta página se muestra un circuito que puede calcular el complemento a 2 de un número de 8 bits de la forma descrita. Con el símbolo CA se ha indicado el circuito que proporciona la inversión de los bits del número binario a complementar.
La operación de complementación, además, está subordinada a la presencia de la señal C. Si es igual a 1, el número binario se complementa; de otra forma queda sin alteración. Esta señal se utiliza para controlar el funcionamiento del circuito, que tanto puede realizar la suma como la resta. En la página siguiente, arriba, se muestra un esquema que puede controlar las operaciones de suma y resta entre dos operandos según el estado (1, 0) de la línea C. Si se pide la suma, la señal de complementación (C) es igual a 0, el segundo operando queda sin alteración y se calcula una suma. En cambio, si se pide la resta, la señal de complementación es igual a 1, y el primer operando se suma al complemento a 2 del segundo operando.

Multiplicación binaria. Las reglas de la multiplicación binaria son idénticas a las de la multiplicación decimal, como puede deducirse del ejemplo indicado a la derecha.
Cada producto parcial es igual al multiplicando si el multiplicador es 1, o bien es igual a 0 si el correspondiente bit del multiplicador es 0.

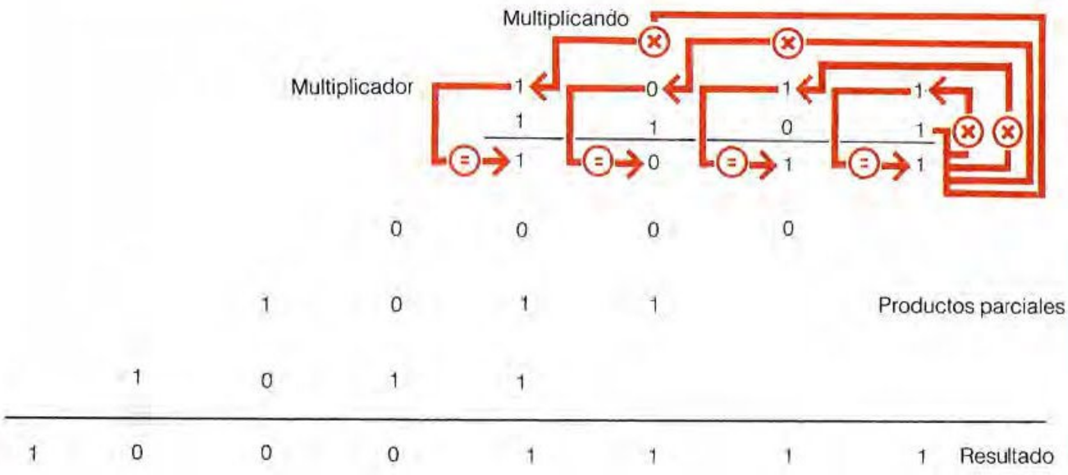
CIRCUITO COMPLEMENTADOR A 2 PARALELO



ESQUEMA DE UN SUMADOR RESTADOR

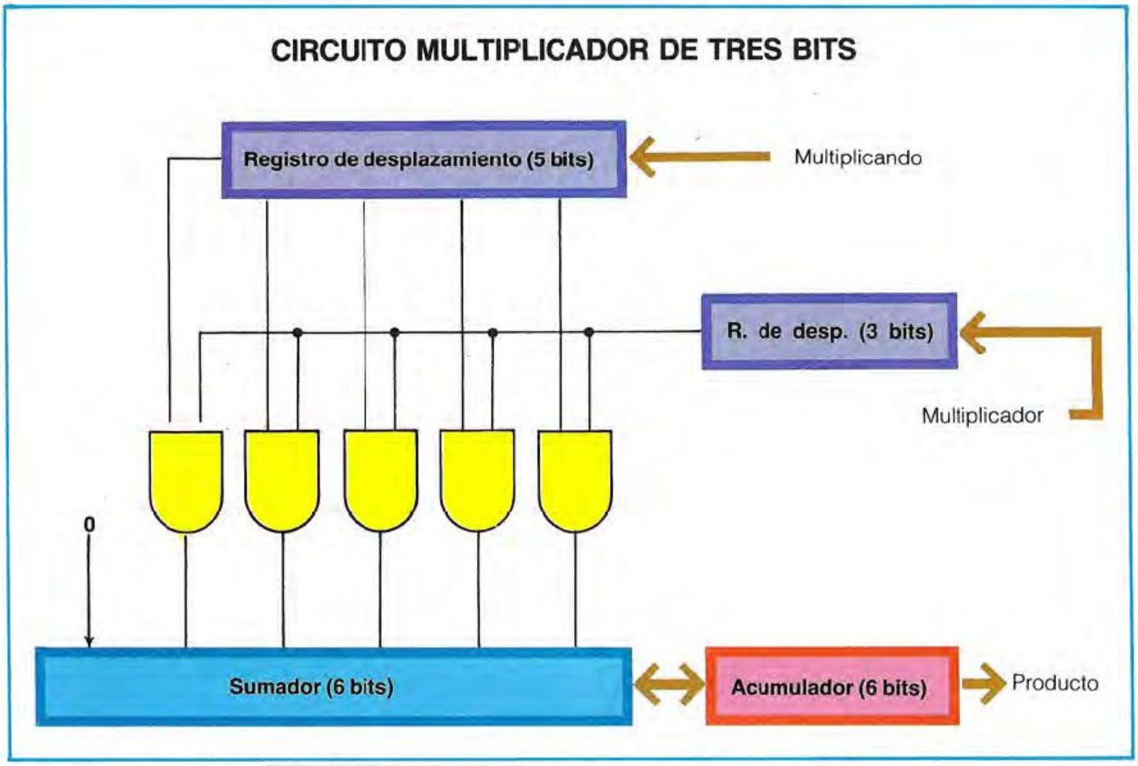


MULTIPLICACION BINARIA



Los productos parciales calculados deben situarse y sumarse oportunamente. Vemos que el producto está compuesto por un número de bits igual a la suma del número de los bits de los operandos. En este caso, el producto tiene una longitud de 8 bits, porque el multiplicando y el multiplicador tienen 4 bits cada uno. En la pág. 884 se muestra un circuito que puede efectuar la operación de multiplicación de dos operandos de 3 bits cada uno. Está compuesto por dos circuitos llamados **registros de desplazamiento** (shift registers), por un sumador de 6 bits, por un acumulador de 6 bits y por una serie de cinco circuitos AND.

Normalmente, el circuito multiplicador no está inserto en la ALU de los microprocesadores utilizados en los ordenadores personales por motivos de sencillez y de costo. En estos casos, la multiplicación se realiza con el método de las sumas repetidas.



Un momento de los trabajos de conexión y verificación de los enlaces exteriores.



A. Tiracchia/AFE

Con este sistema se calcula la multiplicación sumando el multiplicando un número de veces igual al multiplicador. Es decir, la multiplicación 8 × 3 es equivalente a 8 + 8 + 8. En otros tipos de máquina se han previsto circuitos similares al indicado para realizar la multiplicación por hardware y, por tanto, mucho más rápidamente.

División binaria. La división binaria puede obtenerse con un método similar al utilizado para la multiplicación.
Es posible realizar circuitos especializados para la división, pero normalmente no están insertos en la ALU, y la división se calcula con el método de las restas sucesivas.

Además de las aplicaciones aritméticas, la unidad lógico-aritmética puede realizar las operaciones booleanas y de desplazamiento. Las operaciones booleanas son operaciones lógicas realizadas sobre datos binarios según las reglas de la aritmética de Boole*.
Las operaciones de desplazamiento son opera-

* George Boole (1815-1864) está considerado unánimemente como el padre de la aritmética de los calculadores. Por primera vez formuló un conjunto de reglas de tipo algebraico que permitían realizar las cuatro operaciones utilizando las funciones lógicas AND, OR, NOT.

ciones de desplazamiento de los bits sencillos que componen una palabra binaria.

Estas operaciones son muy utilizadas en las programaciones en lenguaje Assembler, puesto que permiten realizar de forma sencilla operaciones de análisis y de comparación de los bits sencillos de una palabra binaria.

Operaciones booleanas. Las operaciones lógicas que se realizan más habitualmente en el interior de la ALU son operaciones AND, OR, NOT y XOR (OR exclusiva).

Estas operaciones ya se han descrito (pág. 72 y siguientes), y se ha visto cómo pueden representarse utilizando la tabla de verdad. Para mayor comodidad del lector, en el primer gráfico de abajo se han representado las tablas de verdad de las funciones AND, OR, XOR, NOT. Suponiendo que la longitud de palabra de la ALU sea de 8 bits, la operación de AND entre dos palabras binarias se realiza efectuando la AND entre los bits correspondientes de las dos palabras. Por ejemplo, la operación AND entre las dos palabras binarias 0 1 1 1 0 1 1 0 y 1 1 1 0 0 1 0 1 se obtiene como se indica en el segundo gráfico de abajo, donde los bits menos significativos de las dos palabras (LSB, los símbolos binarios de más a la derecha) se han escrito en la primera línea y los bits más significativos (MSB, los de más a la izquierda) en la última. El circuito que proporciona la AND de dos palabras binarias de 8 bits se muestra en la pág. 886.

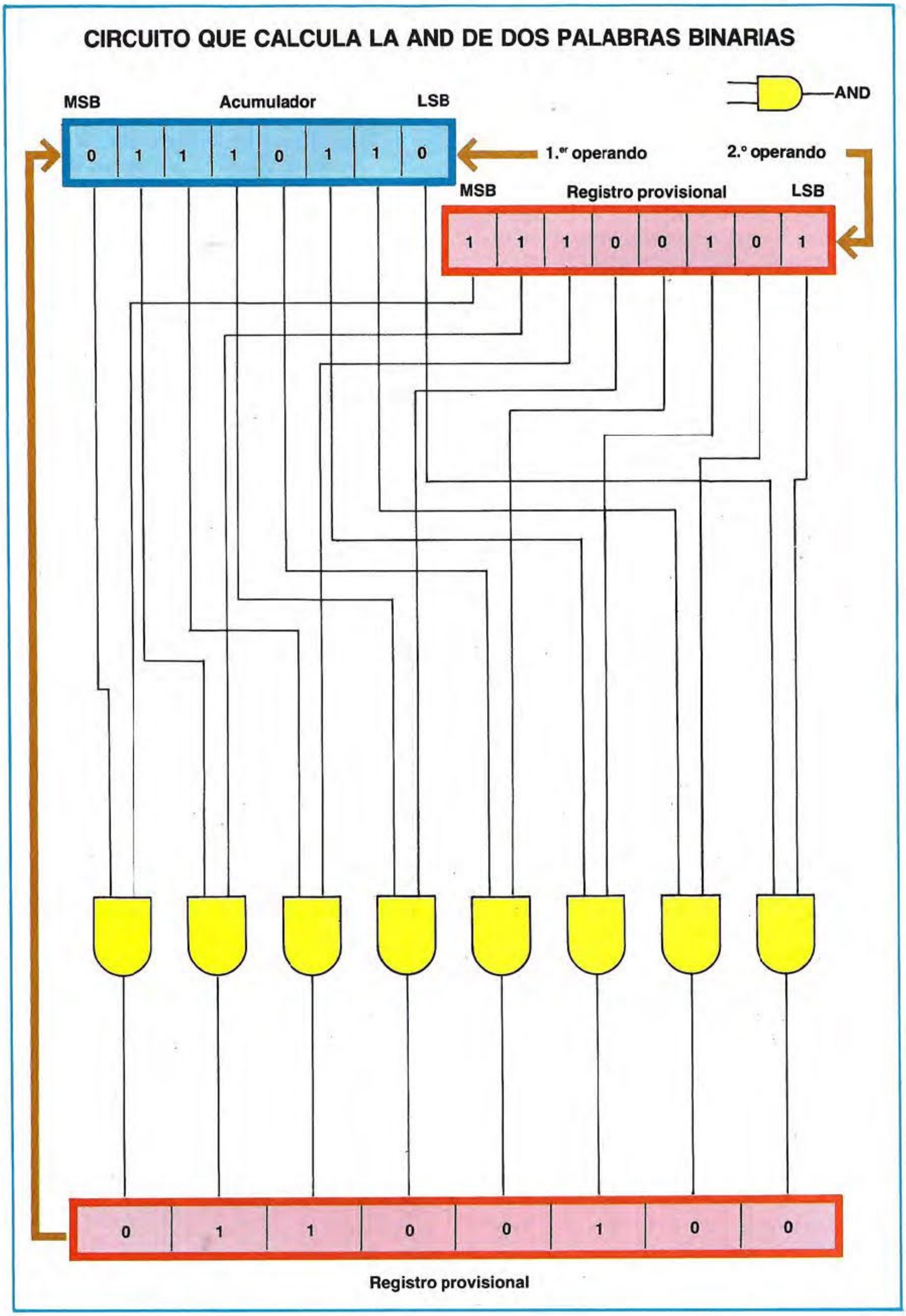
En el esquema se utilizan un registro acumulador para memorizar el primer operando y un registro provisional para memorizar el segundo operando. El resultado de la operación AND se memoriza en un tercer registro, y después se envía al acumulador.

TABLA DE VERDAD DE LAS FUNCIONES LOGICAS AND, OR, XOR, NOT

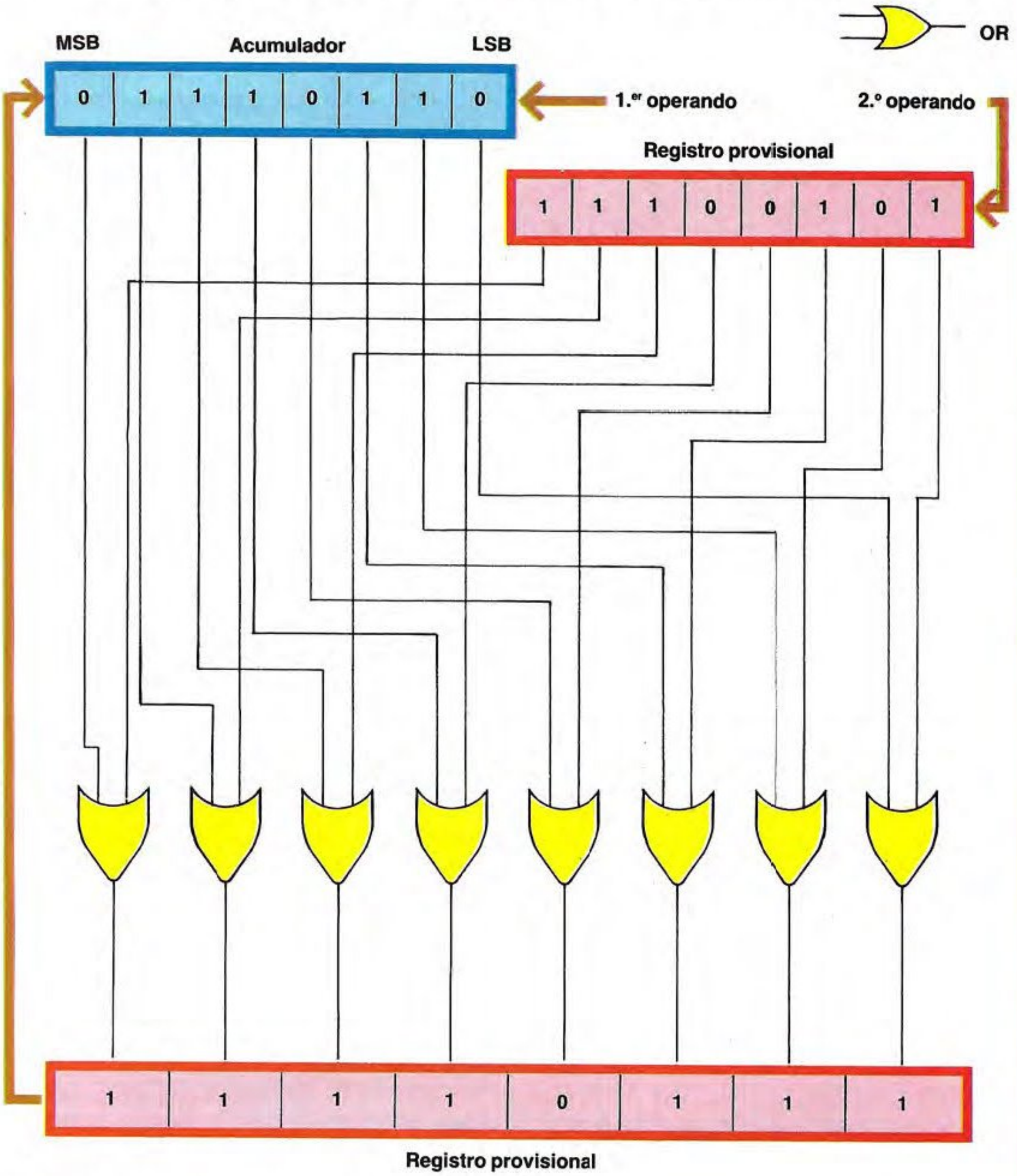
A	B	AND	OR	XOR	NOTA
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

AND DE DOS PALABRAS BINARIAS

(LSB)	0		1		0	(LSB)
	1		0		0	
	1		1		1	
	0	AND	0	=	0	
	1		0		0	
	1		1		1	
	1		1		1	
(MSB)	0		1		0	(MSB)


CIRCUITO QUE CALCULA LA AND DE DOS PALABRAS BINARIAS
AND
MSB
Acumulador
LSB
0 1 1 1 0 1 1 0
1.er operando
2.° operando
MSB
Registro provisional
LSB
1 1 1 0 0 1 0 1
0 1 1 0 0 1 0 0
Registro provisional

CIRCUITO QUE CALCULA LA OR DE DOS PALABRAS BINARIAS

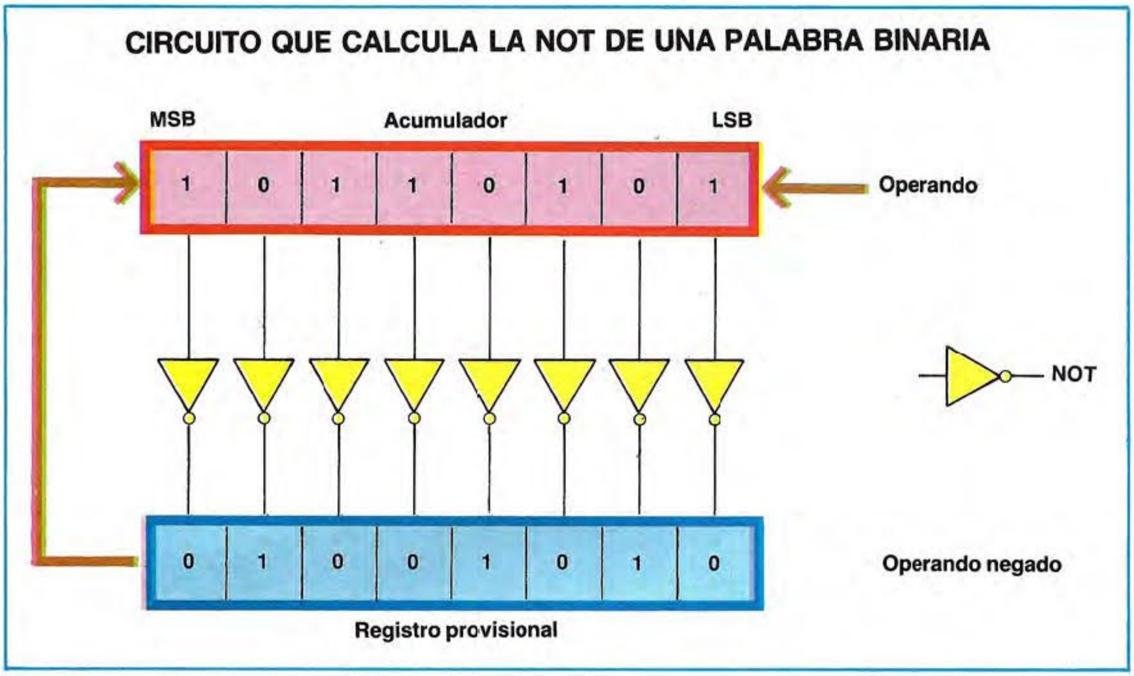


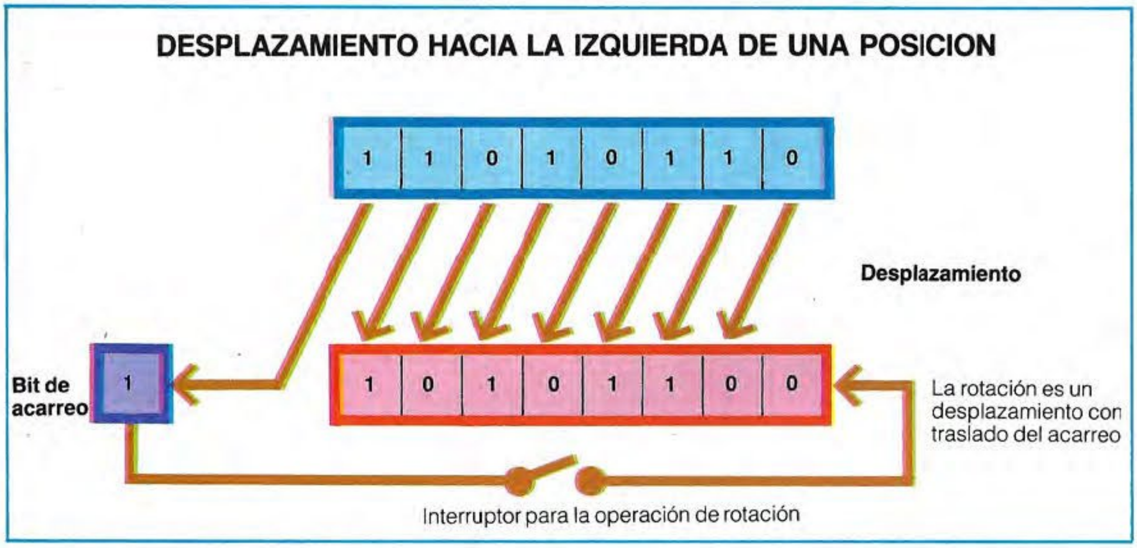
En cambio, la operación OR puede obtenerse como se muestra arriba, donde los operadores AND se han sustituido por operadores OR.

Para el cálculo de las funciones lógicas no es necesario insertar en la ALU todos los circuitos correspondientes. Normalmente sólo se han previsto circuitos de dos tipos: o el par NOT y AND o bien el par NOT y OR. Esto es porque cualquier función booleana puede realizarse con estos pares de operadores. Por ejemplo, la función booleana A OR B es equivalente a

NOT [(NOTA) AND (NOTB)]

Por tanto, para realizar la OR de dos palabras binarias A y B, basta con negar las dos palabras (NOTA, NOTB), calcular la AND y después negar el resultado.



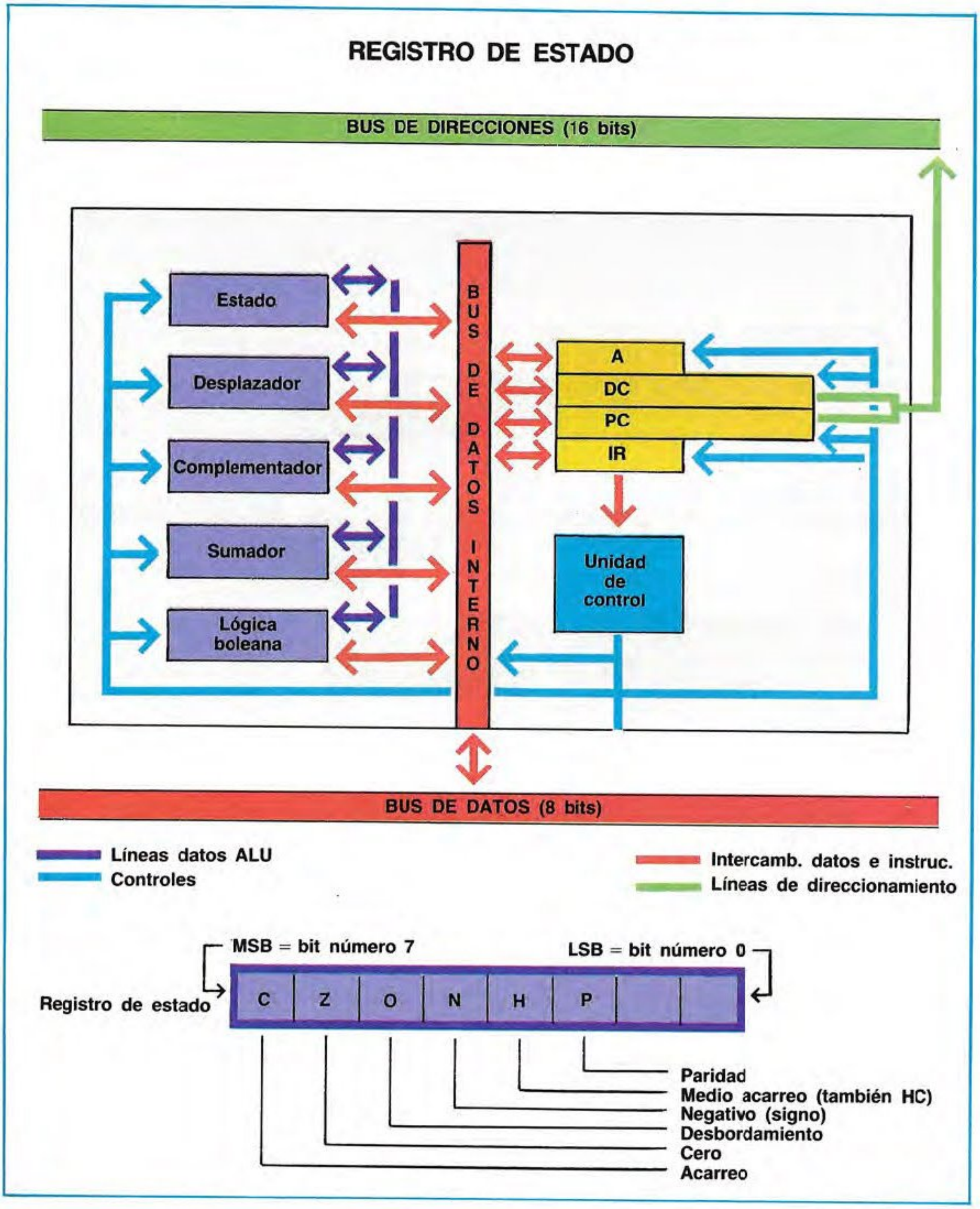


Arriba se ha representado el esquema de un circuito que calcula la función NOT.

Operaciones de desplazamiento. En las operaciones de desplazamiento, los símbolos (0, 1) de la palabra binaria se hacen correr hacia la derecha o hacia la izquierda. Arriba se ha mostrado el desplazamiento de una palabra binaria hacia la izquierda. Generalmente, el bit más significativo después del desplazamiento, se pone en un bit de acarreo (bit de carry).
Además de la operación de desplazamiento, puede ser interesante rotar la palabra; en este caso, el contenido del bit de acarreo se inserta en el bit menos significativo de la palabra binaria resultante después del desplazamiento.
Las operaciones de desplazamiento y de rotación pueden realizarse un número de veces prefijado tanto hacia la izquierda como hacia la derecha. Desplazar un puesto hacia la izquierda equivale a multiplicar por 2, y un puesto hacia la derecha equivale a dividir por 2. Más en general, el desplazamiento de n posiciones hacia la izquierda equivale a la multiplicación de 2^n, mientras que desplazar n posiciones hacia la derecha significa dividir por 2^n.

El registro de estado

En la unidad lógico-aritmética, la ejecución de las operaciones hace indispensable el uso de indicadores que avisen si se ha verificado o no una cierta condición.
Estos indicadores, llamados normalmente flags, están reagrupados en un registro de la ALU. Cada uno de ellos está representado por uno de los bits de este registro y asume el valor 1 o 0 según que la condición bajo control se verifique o no. El registro que contiene los flags se llama **registro de estado** (Status Register); el número de bits que lo constituye varía con el tipo del microprocesador (ver gráfico de abajo).



Normalmente, en el registro de estado hay los siguientes flags:

- Acarreo (Carry)
- Cero (Zero)
- Desbordamiento (Overflow)
- Negativo (Negative o Sign)
- Medio Acarreo (Half Carry)
- Paridad (Parity)

Acarreo. El flag de acarreo se pone a 1 si la última operación ha generado un acarreo sobre el bit más significativo (MSB). Se utiliza en la suma de números compuestos por más palabras (por longitud de palabra se entiende el número de bits presente en el acumulador).

En las operaciones de desplazamiento, el flag de acarreo se utiliza como si la palabra estuviese compuesta por un bit de más (el bit de acarreo): el bit «descargado» se coloca en el acarreo (ver gráfico de abajo). En algunos microprocesadores, este sistema permite dejar disponible una instrucción que activa el desplazamiento «circular» de los bits utilizando un bit de acarreo (ver gráfico de la página siguiente).

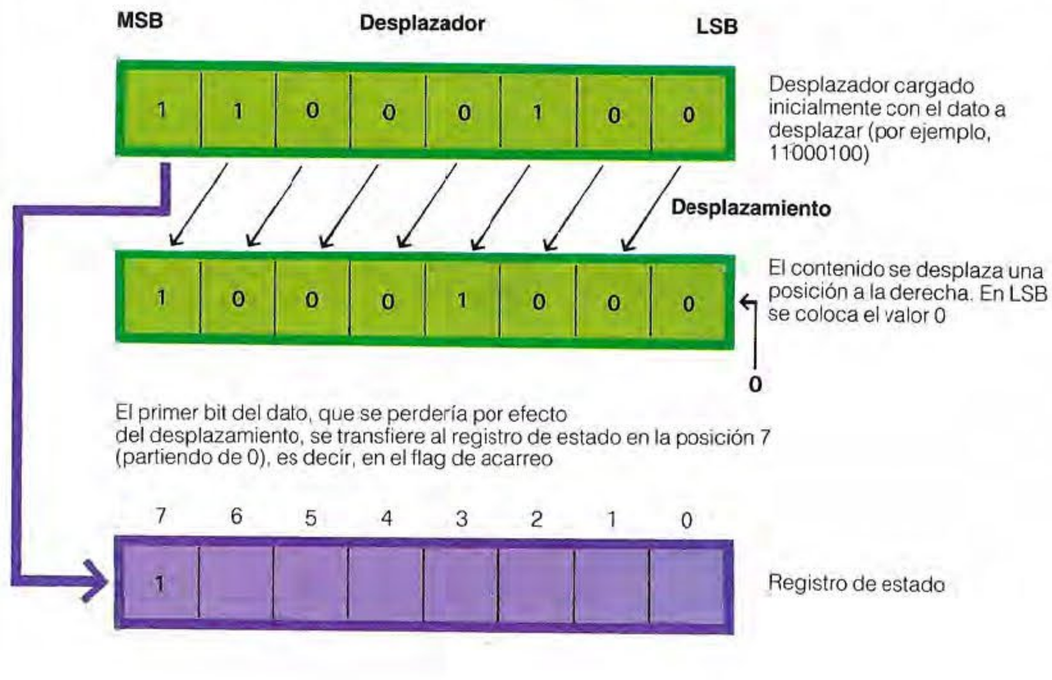
Cero. El flag de cero se pone a 1 si el resultado de la última operación realizada ha sido 0. Es muy utilizado en los programas Assembler para salir de un ciclo (test de cero), o bien en la instrucción de comparación.

Desbordamiento. El flag de desbordamiento asume el valor 1 cuando el resultado de la última operación no es correcto a causa de un acarreo después de la cifra más significativa. Normalmente, el último bit (MSB) es el bit del signo. Esto significa que hay un acarreo sobre el penúltimo bit y, por tanto, el resultado no entra en el espacio disponible. Por ejemplo, si se suma el número $(64)_{10}$ con el número $(96)_{10}$ se obtiene, teniendo en cuenta que el último bit es el del signo (0 si es positivo, 1 si es negativo):

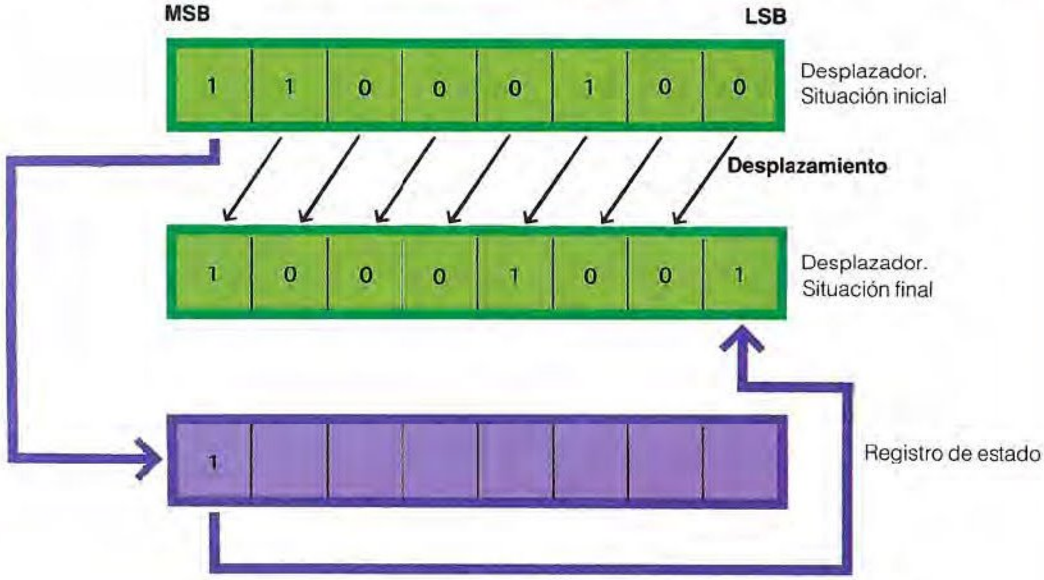
```
Bit del     Bit más
signo     ┌ significativo
 └─► 0 1 0 0 0 0 0 0     (2^6 = 64 decimal)
     0 1 1 0 0 0 0 0     (2^6 + 2^5 = 96 decimal)
     ───────────────
 ┌─► 1 0 1 0 0 0 0 0
```

Acarreo que produce un desbordamiento y el bit 0 de registro de estado se pone a 1.

OPERACION DE DESPLAZAMIENTO CON USO DEL BIT DE ACARREO



ROTACION A LA IZQUIERDA UTILIZANDO EL BIT DE ACARREO



Después del desplazamiento, en el LSB del desplazador se transfiere el contenido del acarreo. Esto equivale a un desplazamiento «circular» del contenido: MSB se convierte en LSB, y todos los demás bits se desplazan una posición a la izquierda

Negativo. Es el flag del signo, y se pone a 1 cuando a la salida de la última operación realizada el bit más significativo (MSB) vale 1.
Este bit también se llama «bit negativo», puesto que en la notación en complemento a 2, el MSB indica el signo con la convención:

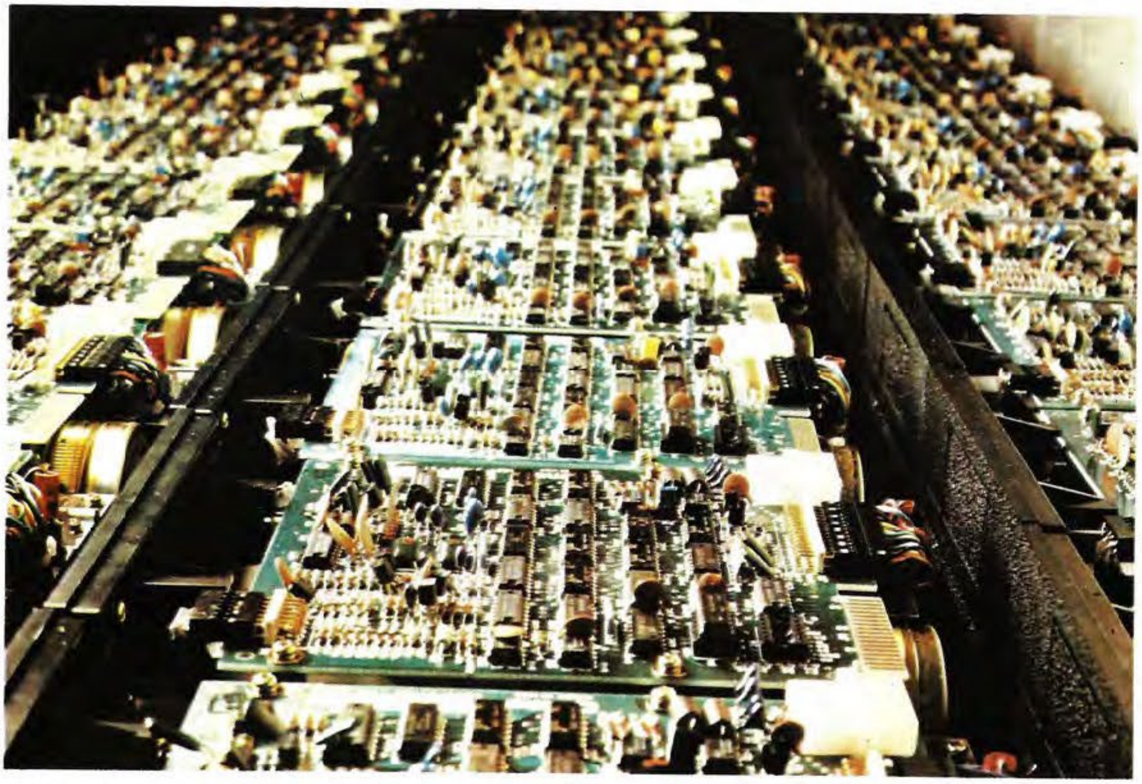
0 = signo positivo
1 = signo negativo

El bit del signo se utiliza para realizar controles sobre el signo del resultado de una operación (positivo o negativo).

Medio acarreo. El flag de medio acarreo se utiliza exclusivamente en los microprocesadores de 8 bits que trabajan con aritmética BCD y se pone a 1 si la última operación ha generado un acarreo sobre los primeros 4 bits de la palabra. Para representar cada cifra decimal, el código BCD utiliza 4 bits*, y normalmente en un microprocesador de 8 bits caben dos códigos BCD en cada palabra. En una operación de suma podría tenerse un acarreo entre la cifra BCD menos significativa (primeros 4 bits) y la siguiente (últimos 4 bits). En este caso, el bit de medio acarreo (o intermediate carry) se pone a 1.

Paridad. El flag de paridad se pone a 1 si, en el resultado de una operación, el número de bits que tienen valor 1 es par (paridad par). También existe la posibilidad de tener la paridad impar: el bit, o flag, de paridad se pone a 1 cuando el número de bits de valor 1 es impar.
El control de paridad se utiliza en la manipulación de los caracteres o en la transmisión a los dispositivos externos. En este último caso es importante saber si cada carácter ha sido recibido correctamente. El control se realiza en el dispositivo receptor.
Por ejemplo, en caso de error por inversión de un bit (un bit a 1 pasa a ser 0, o viceversa) el control de paridad resulta erróneo, y se pide la retransmisión del carácter. Sin embargo, este sistema no ofrece un elevado grado de seguridad, puesto que si se verifica una doble inversión, el control de paridad no evidencia el error.

* El código BCD está compuesto por 4 bits con su significado normal de potencias de 2 (8, 4, 2, 1) y excluye todas las configuraciones que darían un valor decimal superior a 9. Recuérdese que con 4 bits se pueden escribir números binarios hasta el valor 8 + 4 + 2 + 1 = 15 (decimal).



J. Pickerell/Marka

Línea de montaje de placas de ordenador.

Normalmente, en los intercambios de datos en el interior de un microprocesador no se realizan controles de paridad.

Flags particulares. En algunos microprocesadores existen otros flags, como por ejemplo el flag de habilitación de la interrupción (interrupt enable), que se pone a 1 cuando es posible interrumpir la función que la CPU está ejecutando. En el caso de que un dispositivo tenga necesidad de ser servido envía, sobre una adecuada línea, una señal de interrupción a la CPU. Esta señal sólo será atendida si el bit de interrupción está a 1. En otras palabras, la unidad de control puede servir un dispositivo externo sólo cuando existe una habilitación a las interrupciones.
Si al producirse una petición de interrupción el flag de interrupción está habilitado, la unidad de control manda una serie de señales que envían a ejecución una rutina de servicio.
Esta rutina suspende el programa que está en ejecución en aquel momento, salvando el contenido de los registros (contador de programa, acumulador, etc.) en la pila (stack). En base al tipo de interrupción se realiza un programa de servicio específico y, al final, atendida la petición del dispositivo externo, se reemprende la ejecución del programa suspendido anteriormente. El programador tiene facultad de deshabilitar las interrupciones. En algunos casos, zonas de programa particularmente críticas no pueden interrumpirse: entonces la interrupción se deshabilita, y las eventuales peticiones se atienden sólo al final de estas fases. También la CPU puede deshabilitar automáticamente las interrupciones, y esto sucede, por ejemplo, en la inicialización del sistema o cuando ya está en curso una interrupción.
Es muy importante tener presente que en un microprocesador, los flags de estado no siempre están a cero antes de la ejecución de un ciclo de instrucción.
Esto significa que pueden tenerse flags a 1 no porque hayan sido puestos en este estado por la instrucción en curso, sino como resultado de una instrucción anterior. Por tanto, es necesario que al escribir un programa Assembler, el programador sepa correctamente, antes de efectuar un test, qué flags han intervenido en las diversas instrucciones.

El buffer de la ALU

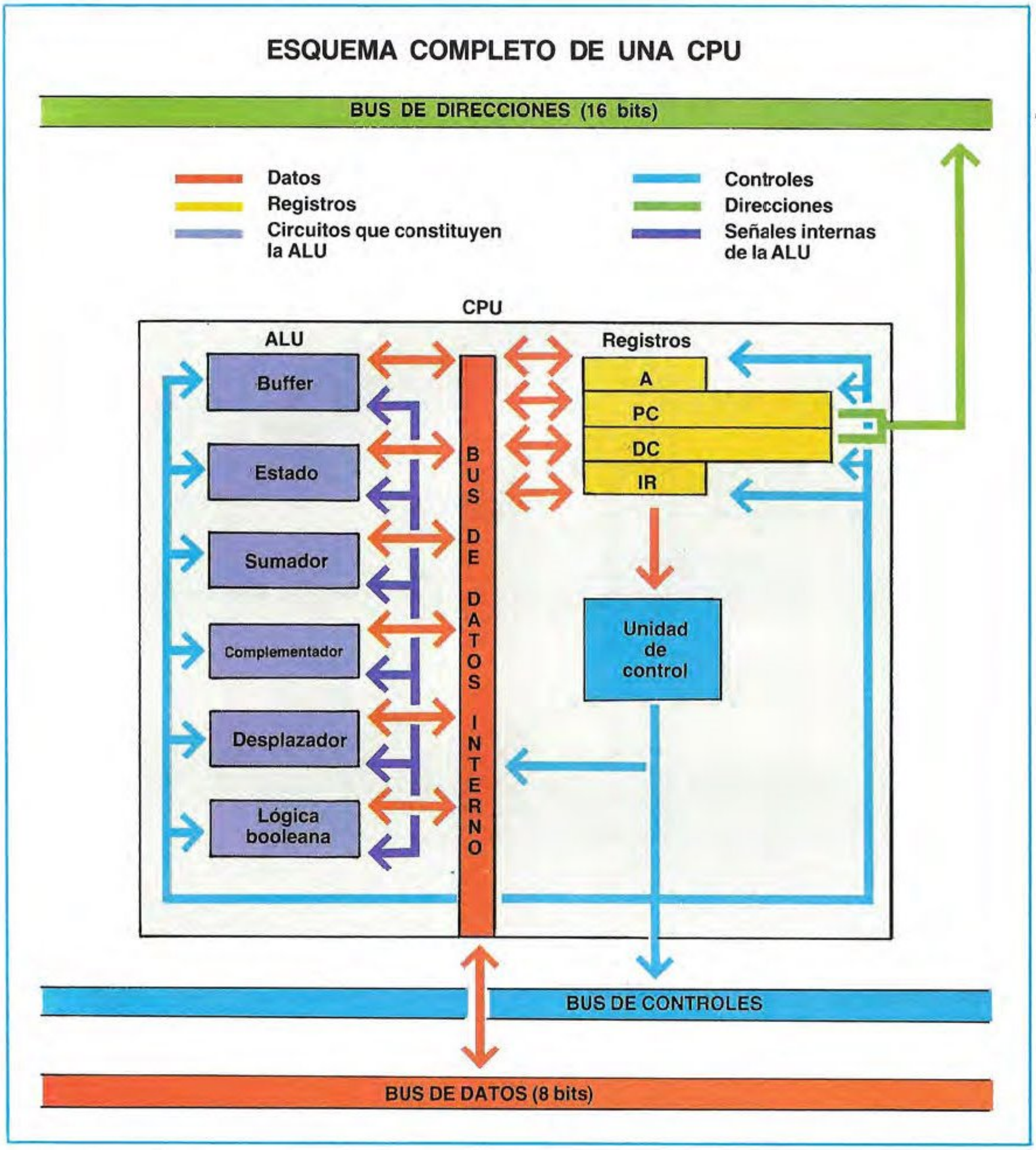
Un último elemento presente en la ALU es un registro acumulador de caracteres provisional, utilizado como memoria de almacenamiento durante los cálculos (gráfico de abajo). Por ejemplo, al sumar dos números, el primero de ellos se deposita en el acumulador; el segundo (en el ciclo de instrucción siguiente) se memoriza provisionalmente en este registro buffer. Los circuitos ALU proceden a realizar la suma de los contenidos del acumulador y del buffer, depositando el resultado en el acumulador.

La unidad de control

La existencia en la ALU de los diversos dispositivos que permiten la realización de las operaciones aritméticas, booleanas y de desplazamiento no es suficiente para garantizar el correcto funcionamiento del procesador.

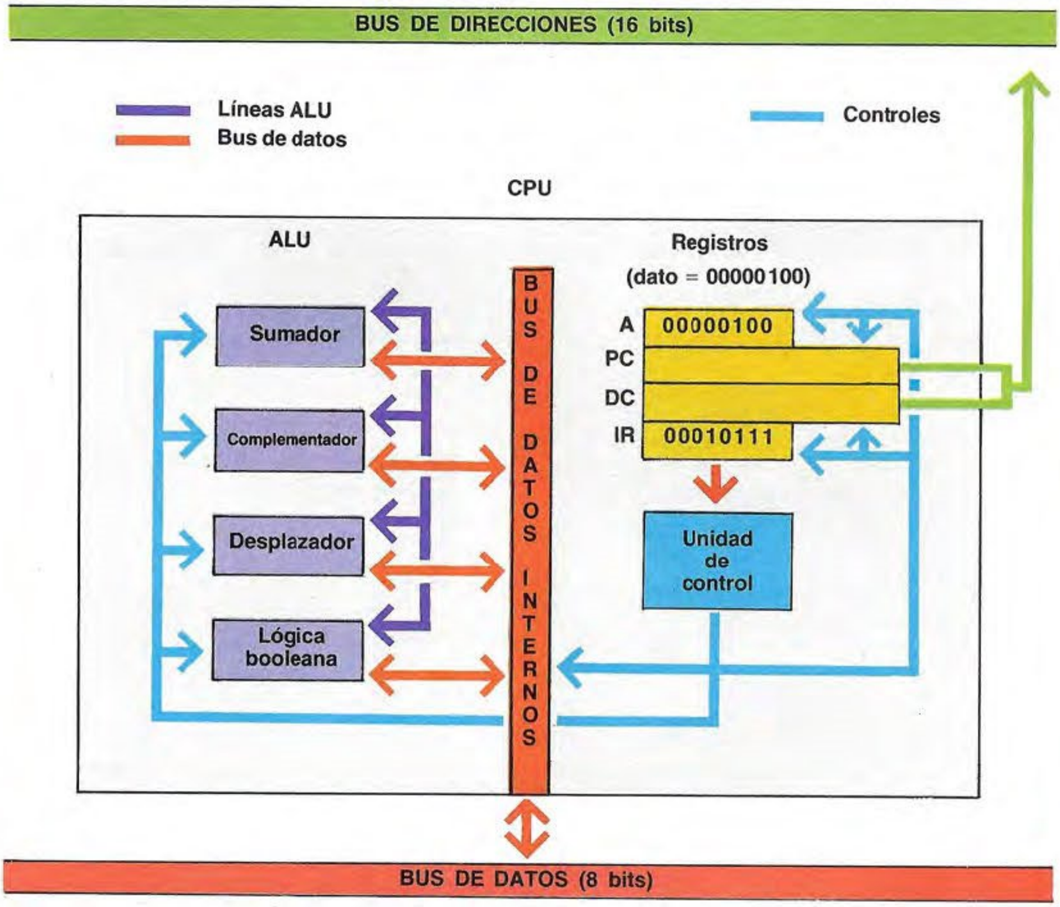
Todas las operaciones que la unidad lógico-aritmética puede realizar están gestionadas por la unidad de control.

El primer paso de la secuencia de ejecución de una instrucción consiste en la descodificación del contenido del registro de instrucciones.

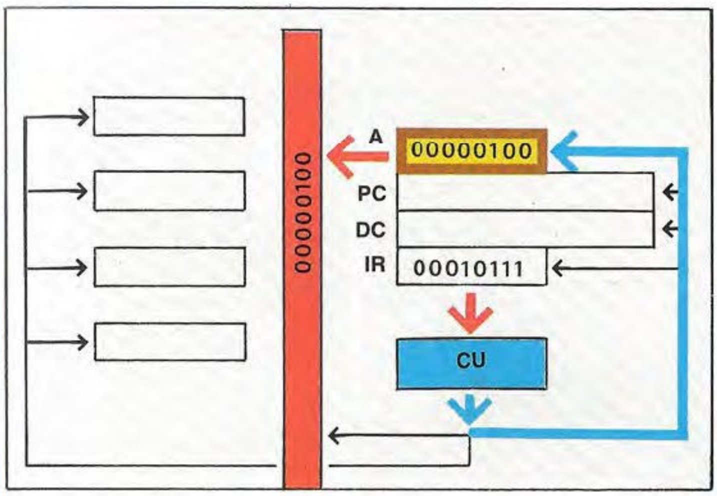


En base al tipo de instrucción, o sea según el código binario contenido en IR, la unidad de control genera una secuencia de señales de habilitación que permite a los elementos de la ALU recibir los datos en el preciso instante en que son llamados para realizar sus funciones.
En las págs. 894 a 896 se ilustra la ejecución de una instrucción de desplazamiento a la izquierda del contenido del acumulador.
Todos los dispositivos internos de un microprocesador intercambian informaciones entre sí y con el exterior sirviéndose de las mismas líneas. La unidad de control (CU, Control Unit) tiene la misión de permitir el acceso al bus a un solo dispositivo cada vez y desconectar en el mismo instante todos los demás. Los dispositivos conectados al bus están provistos de circuitos de entrada/salida particulares, llamados «de tres estados» (three-states), que se conectan al bus sólo con la llegada de una señal de habilitación. Este procedimiento también se aplica a los dispositivos conectados a los bus externos (circuitos I/O, memorias, etc.). En este caso, en los chips se ha previsto una patilla de selección, llamado Chip Enable o Chip Select, conectada a una línea de control exterior. Bajo el control del microprocesador, esta línea permite habilitar uno de los dispositivos y acceder al bus, al mismo tiempo que se deshabilitan todos los demás (ver gráfico de la pág. 897).

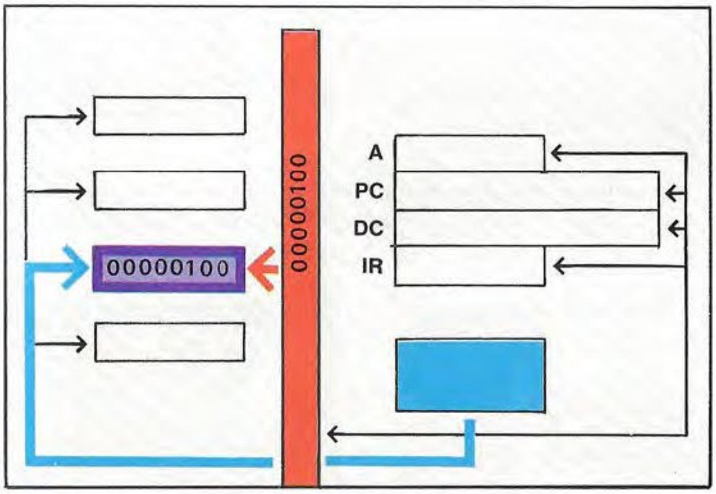
EJECUCION DE UNA INSTRUCCION DE DESPLAZAMIENTO A LA IZQUIERDA EN LA CPU



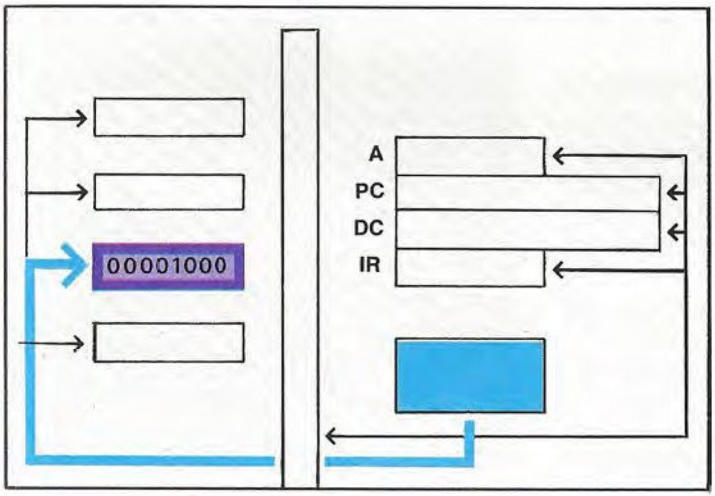
1 Estado inicial de la CPU. El acumulador A contiene el dato a desplazar, y el registro de instrucciones IR contiene el código de la instrucción a realizar (desplazamiento a la izquierda)



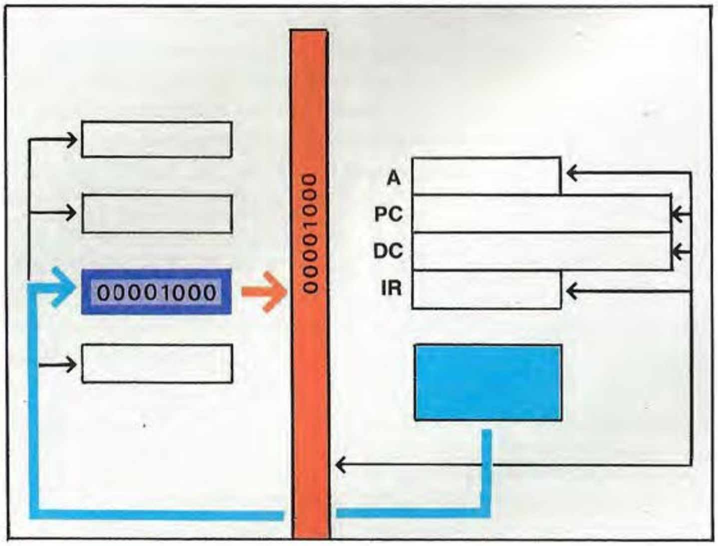
2 La unidad de control descodifica y reconoce la instrucción. La primera señal de control activa la transferencia del contenido del acumulador al bus de datos interno



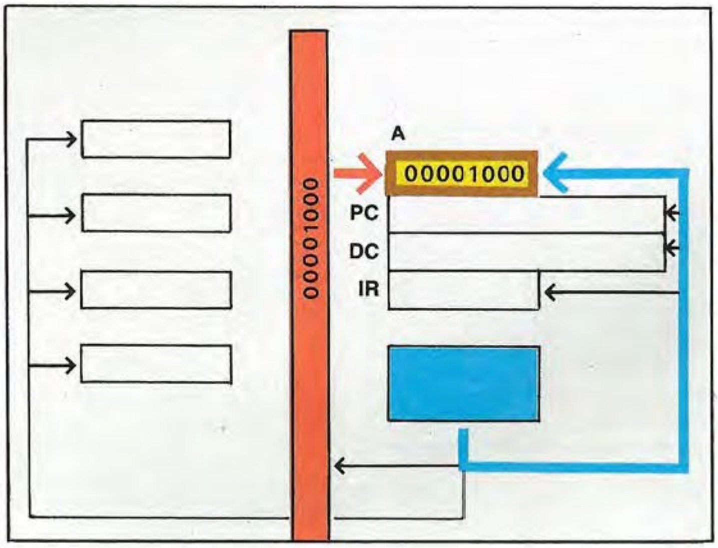
3 La siguiente señal de control activa la transferencia del dato desde el bus al circuito de desplazamiento (interno a la ALU)



4 A continuación del envío de una nueva señal por parte de la unidad de control, el contenido del circuito de desplazamiento se desplaza una posición a la izquierda; la última posición a la derecha (LSB, bit menos significativo) se iguala a cero



5 Al final de la operación de desplazamiento, el contenido del desplazador se transfiere al bus de datos interno



6 El dato se transfiere del bus al acumulador, donde sustituye el valor anterior (dato de partida). En el acumulador ahora hay el valor inicial (00000100) escalado en una posición hacia la izquierda (00001000)

Ejecución de las instrucciones

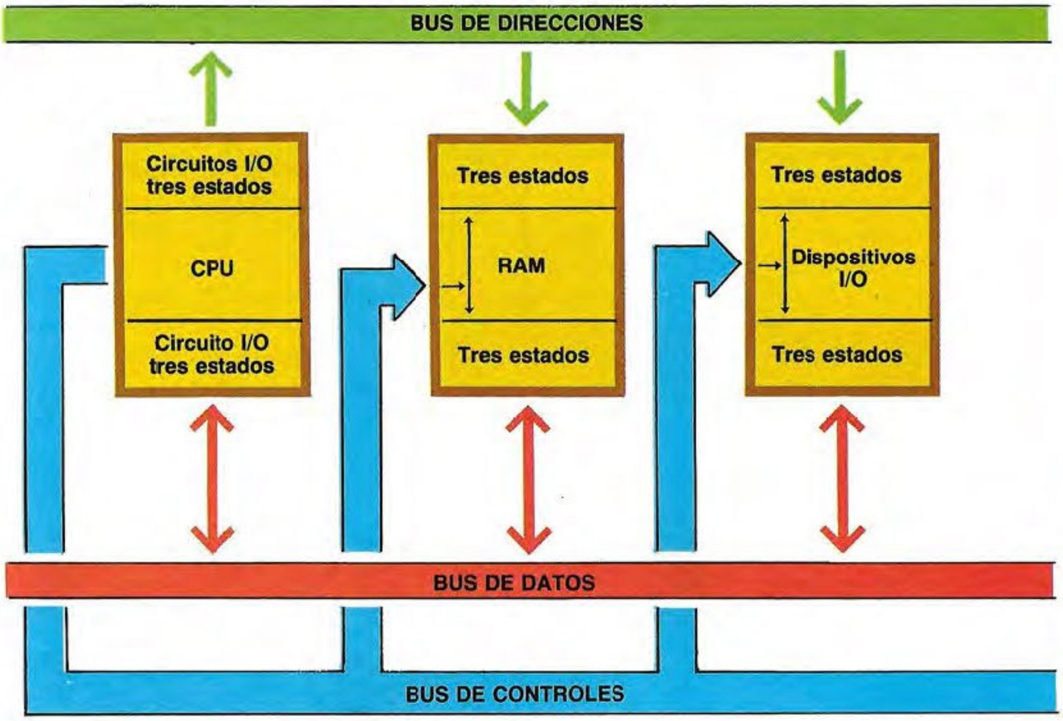
La fase de ejecución de las instrucciones es posiblemente la que utiliza en mayor grado los resortes internos de un microprocesador, y también es la que más rehuye un tratamiento claro y orgánico.
Por tanto, la lectura de los párrafos que siguen necesitará una atención particular: la comprensión de los mecanismos internos de gestión de la CPU facilitará enormemente el estudio de las instrucciones del lenguaje Assembler, que están estrechamente ligadas al funcionamiento paso por paso de la máquina.

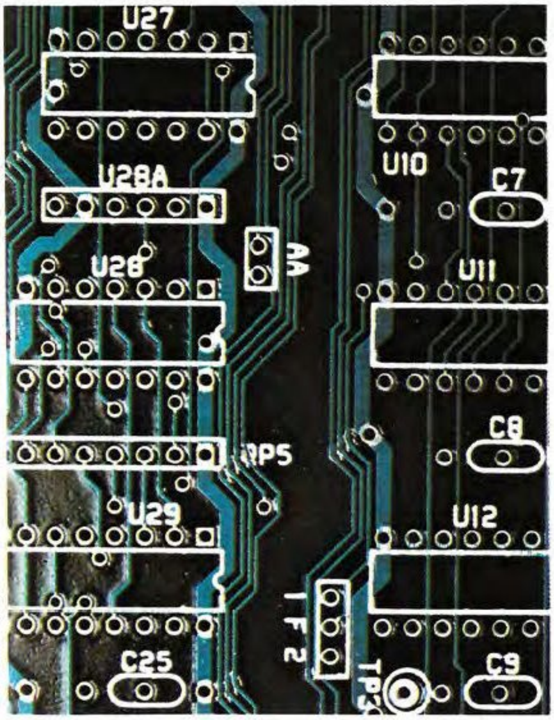
La temporización de las señales

En el gráfico de la pág. 898 se muestran las líneas de conexión que gestiona la CPU, que son:

- las líneas de dirección de 16 pistas. Cada bit viaja sobre una pista, y así pueden direccionarse 65535 elementos de memoria
- las líneas de datos, de 8 pistas. Es posible transferir al mismo tiempo un dato compuesto de 8 bits
- las líneas de control: conjunto de pistas que habilitan de forma unívoca el tipo de opera-

USO DE LOS CIRCUITOS TRES ESTADOS EN UN SISTEMA DE MICROPROCESADOR





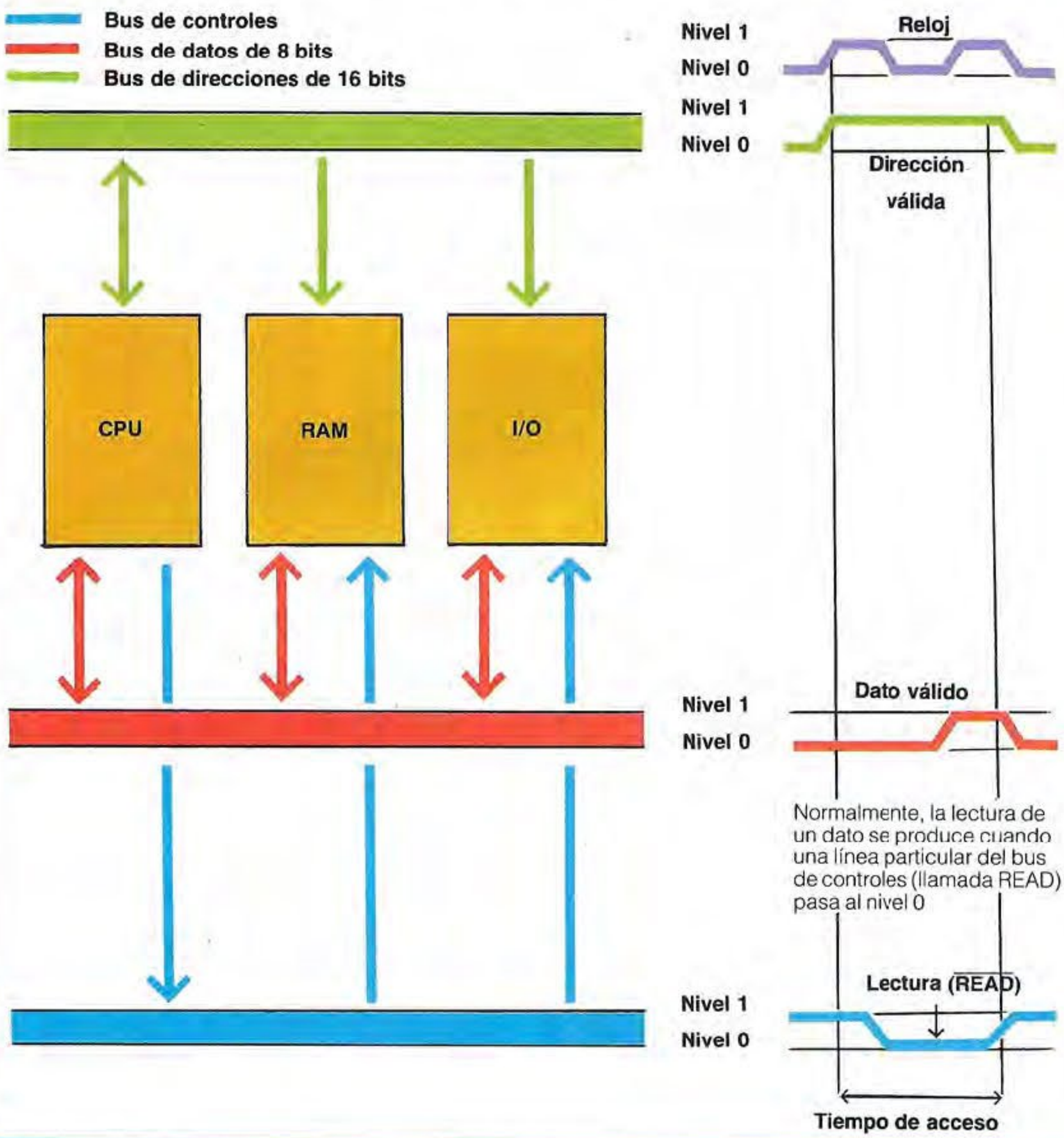
Detalle de las pistas de una tarjeta.

ción a realizar. La disposición y el número de estas líneas dependen esencialmente del tipo de microprocesador.

Para que un sistema de microprocesador pueda trabajar correctamente deben respetarse algunas reglas precisas de temporización de las señales intercambiadas entre los diversos dispositivos. Por este motivo, la transferencia en el bus de direcciones, en el bus de datos y en el bus de controles está ligada a precisas relaciones temporales. En la pág. 898 se ha ilustrado un diagrama de secuencia de las señales necesarias para la realización de una operación de lectura en memoria.

El proceso empieza con la transferencia del contenido del contador de programa al bus de direcciones, sincronizado con el reloj del sistema. Para que la memoria deje disponible en el bus de datos el valor pedido, es necesario que transcurra un cierto número de períodos de reloj, o sea que se verifique en la señal de reloj un cierto número de transiciones de 1 a 0; en este caso, por razones de sencillez, se ha considerado suficiente un solo período.

PROCESO DE LAS SEÑALES QUE GENERAN UNA LECTURA EN MEMORIA



El intervalo de tiempo correspondiente está definido como **tiempo de acceso**, y representa el máximo retardo entre el momento en que se pide un dato y el momento en que este dato queda disponible. Las señales presentes en las líneas de datos, correspondientes al valor a transferir, deben ser estables durante un cierto intervalo de tiempo para que el microprocesador pueda transferirlas a sus registros.

El tipo de operación necesario (en el ejemplo, una lectura) está especificado por las señales presentes en las líneas de control, que habilitan el dispositivo que recibe o transmite el dato. La temporización de las señales es uno de los parámetros más críticos del proyecto de un sistema de microprocesador. Los tiempos de acceso (ver gráfico de la derecha) pueden variar, por ejemplo, en función de la temperatura. Las informaciones presentes en las puertas del microprocesador nunca están perfectamente sincronizadas con el reloj, y la falta de sincronismo produce tiempos de retardo no despreciables. Para tener en cuenta estos retardos, el sistema debe evaluar convenientemente el retardo que

transcurre entre el impulso de reloj y la aparición de las direcciones en el bus (Tda), el tiempo necesario a la memoria para responder y el retardo necesario hasta que el dato se ha establecido en las líneas; las temporizaciones dependen del tipo de operación en curso.

La operación de lectura que debe considerarse es menos compleja que una operación de escritura. Efectivamente, en esta última deben considerarse los siguientes factores (ver gráfico de la pág. 900, arriba):

1 / la longitud del impulso de escritura (Tw, write pulse)

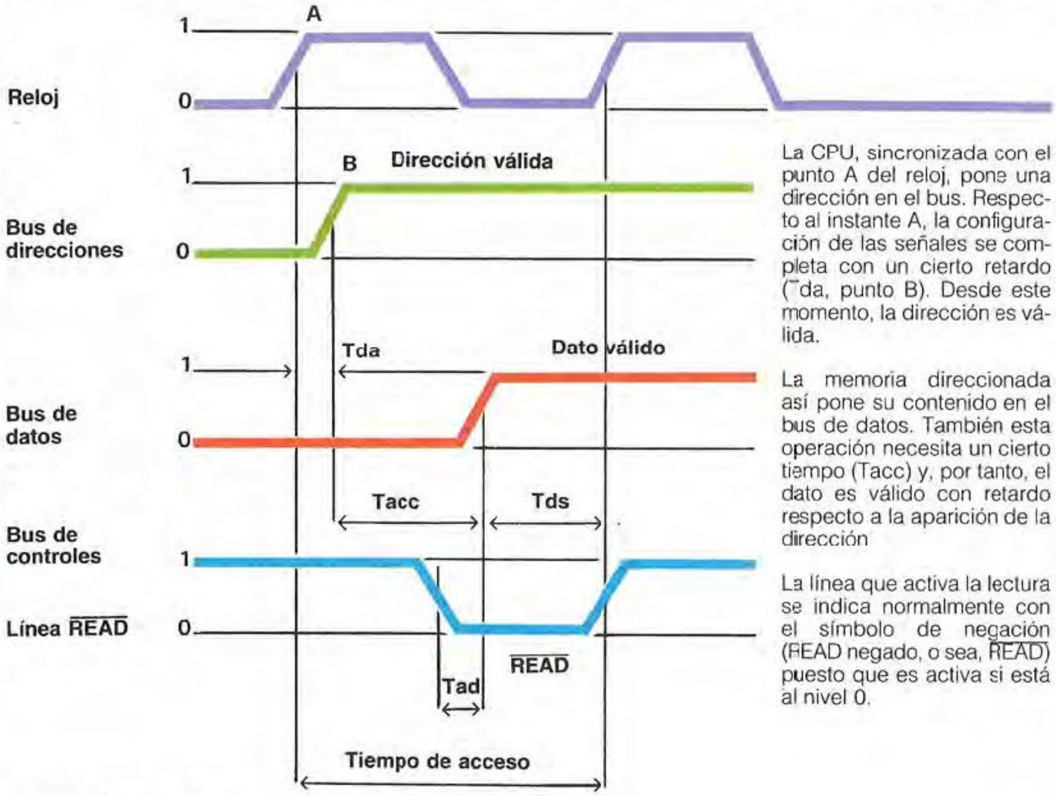
2 / el mínimo intervalo de tiempo que debe esperar antes de enviar el impulso de escritura para que la dirección sea considerada válida (Tas, address setup time)

3 / el mínimo intervalo de tiempo durante el cual el dato es válido antes de que el impulso de escritura termine (Tds, data setup time)

4 / el mínimo intervalo de tiempo durante el cual el dato permanece disponible en las líneas de datos (Tdh, data hold time).

El orden de sucesión de las señales sobre las diversas líneas es crítico, puesto que la CPU introduce, además del retardo en las líneas de dirección (Tda, address delay time), también un retardo en las líneas de datos (Tdd, data delay time) y uno sobre el impulso de escritura (Tdw, write pulse delay time, ver pág. 900, abajo).

RETARDOS EN LAS SEÑALES QUE ACTIVAN UNA OPERACION DE LECTURA



La CPU, sincronizada con el punto A del reloj, pone una dirección en el bus. Respecto al instante A, la configuración de las señales se completa con un cierto retardo (Tda, punto B). Desde este momento, la dirección es válida.

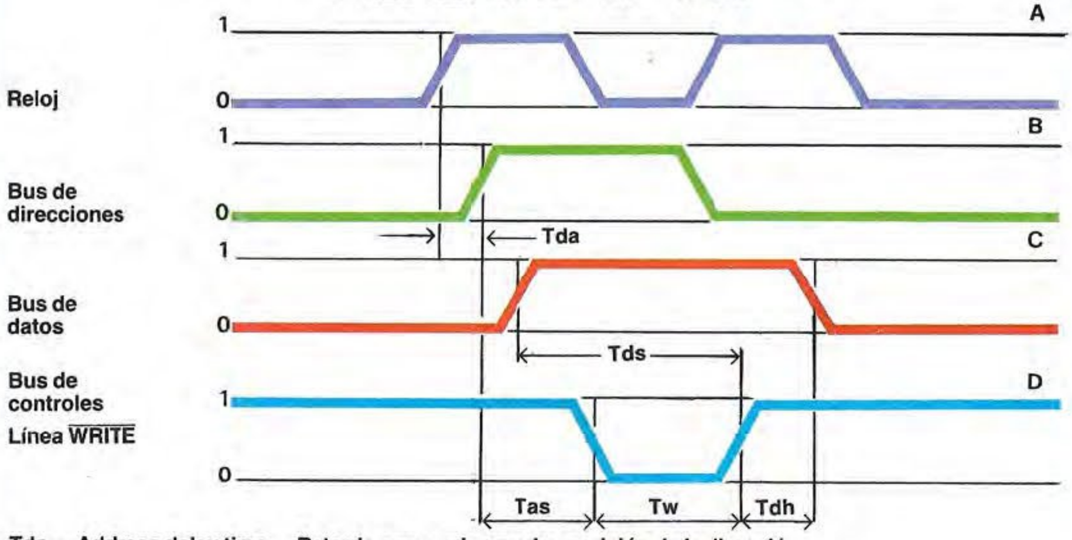
La memoria direccionada así pone su contenido en el bus de datos. También esta operación necesita un cierto tiempo (Tacc) y, por tanto, el dato es válido con retardo respecto a la aparición de la dirección

La línea que activa la lectura se indica normalmente con el símbolo de negación (READ negado, o sea, $\overline{READ}$) puesto que es activa si está al nivel 0.

La línea READ a nivel 1 indica que el dato no está preparado. Transcurrido un cierto tiempo desde la aparición de la dirección (punto B) la línea $\overline{READ}$ se pone al nivel 0 y habilita la operación de lectura

Tda = Address delay time = Tiempo de retardo en las líneas de direcciones
Tacc = Memory access time = Tiempo de acceso a la memoria
Tds = Data setup time = Tiempo necesario para obtener un dato válido
Tad = Data access time = Retardo entre el comando de lectura y la aparición del dato

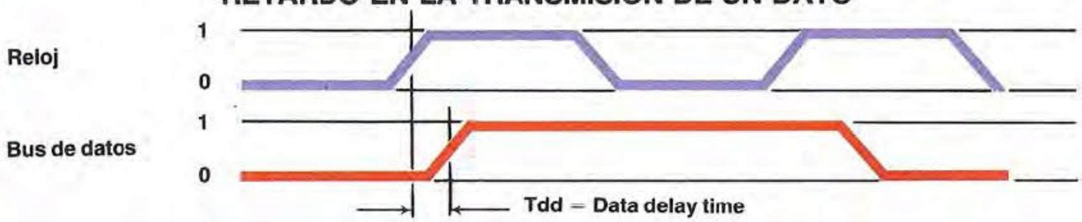
SEÑALES Y TIEMPOS DE RETARDO EN UNA OPERACION DE ESCRITURA EN MEMORIA



Tda = Address delay time = Retardo necesario para la aparición de la dirección
Tds = Data setup time = Tiempo durante el cual el dato puede escribirse
Tas = Address setup time = Tiempo durante el cual puede utilizarse la dirección
Tw = Write pulse = Duración de la señal $\overline{\text{WRITE}}$
Tdh = Data hold time = Tiempo durante el cual hay el dato después del impulso $\overline{\text{WRITE}}$

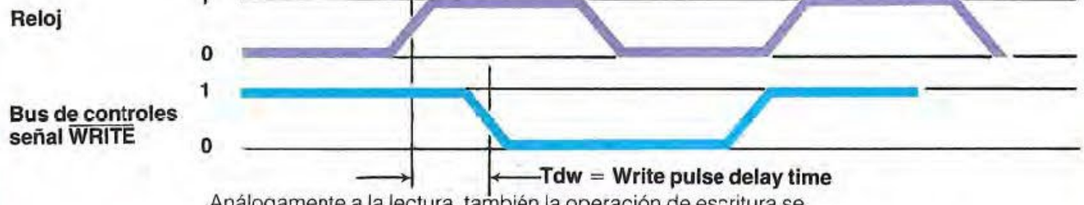
A El reloj activa la operación de escritura
B Las señales en el bus de direcciones son válidas sólo después de un cierto retardo (Tda)
C El dato a escribir aparece con retardo después de la dirección. En este momento, la configuración de las señales está completa, pero el dato todavía no se ha escrito.
D Por último aparece el impulso de escritura ($\overline{\text{WRITE}}$) y la memoria direccionada al punto B adquiere el dato. Al final del $\overline{\text{WRITE}}$, el dato permanece durante el tiempo Tdh, después de que el ciclo ha terminado

RETARDO EN LA TRANSMISION DE UN DATO



Tiempo de retardo entre el reloj y la aparición del dato. La aparición de la señal de reloj (la línea pasa del nivel 0 al nivel 1) activa la función a realizar, pero a causa de los fenómenos físicos presentes, las correspondientes señales eléctricas aparecen sólo después de cierto tiempo. Por tanto se tiene un retardo entre el comando (salida del reloj y la ejecución (aparición de la señal correspondiente al dato)

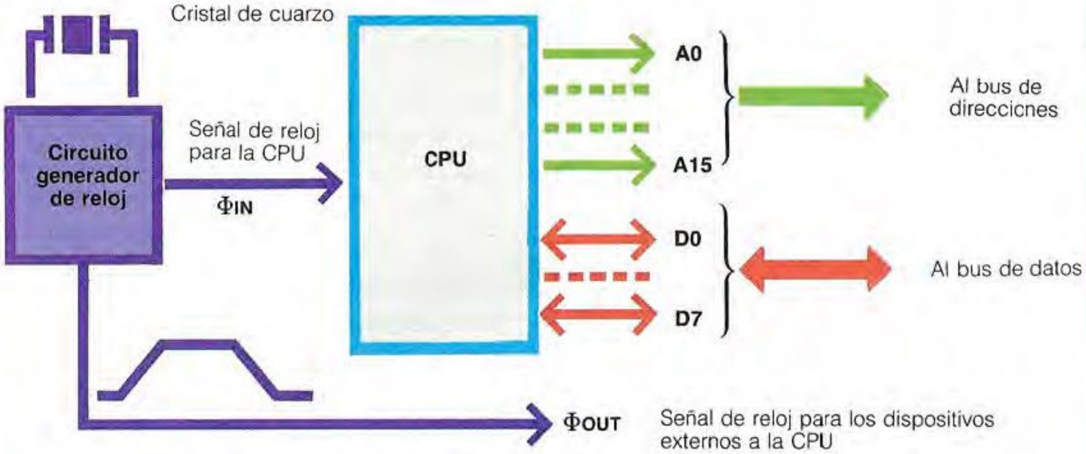
RETARDO EN LAS SEÑALES DE CONTROL



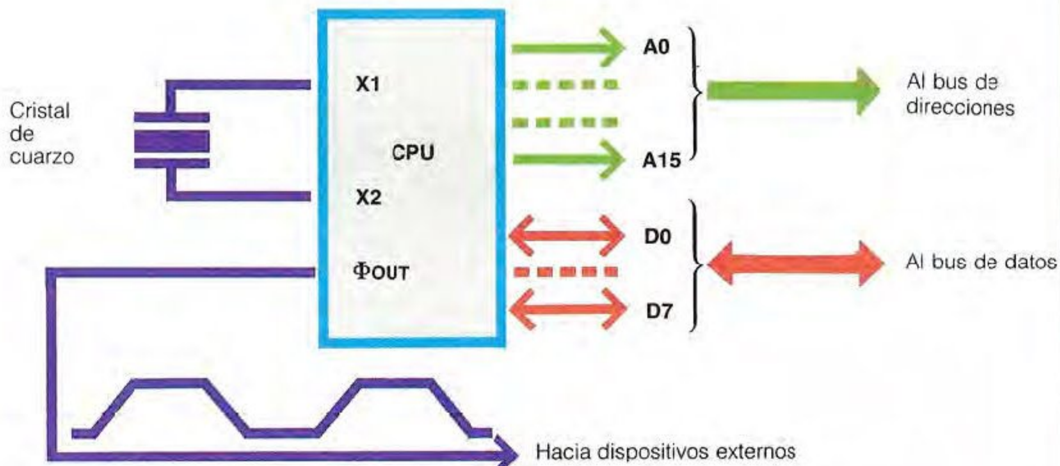
Análogamente a la lectura, también la operación de escritura se activa poniendo a cero el nivel de una particular línea llamada $\overline{\text{WRITE}}$.
El tiempo de retardo de este impulso respecto al de reloj se indica con la sigla Tdw.

Todas estas señales de temporización están sincronizadas con el reloj del sistema.
En algunos microprocesadores, el reloj (representado por el símbolo Φ) está generado por un circuito exterior. Este dispositivo, generalmente constituido por un chip dedicado, contiene un oscilador cuya frecuencia está controlada por un estabilizador de cristal de cuarzo. El primer gráfico de esta página muestra el esquema de una CPU que utiliza un generador de reloj externo. El mismo circuito proporciona la señal de temporización (Φ_{out}) para los dispositivos externos que deben estar sincronizados con la CPU.
En otros microprocesadores, los circuitos de generación de la señal de reloj están integrados en la CPU. En este caso (segundo gráfico), el cristal de cuarzo está conectado directamente a la CPU y la señal de sincronismo (Φ_{out}) está disponible en una patilla del microprocesador. La señal de reloj la utiliza la unidad de control para sincronizar todas las operaciones internas de microprocesador. Durante un ciclo de instruc-

ESQUEMAS DE REALIZACION DE LA SEÑAL DE RELOJ



CPU con oscilador externo. La señal de reloj se genera con un componente externo especializado



Oscilador de reloj integrado en la CPU. En este caso no hay la conexión ΦIN, puesto que la señal correspondiente es interna a la CPU. Además, acompañan a las dos conexiones X1 y X2 para el estabilizador de cristal de cuarzo

ción, los ciclos máquina que lo componen son múltiplos del período de reloj, que también se llama **estado** (ver gráfico de abajo).

Durante estos intervalos, la unidad de control envía las señales necesarias para habilitar los diversos dispositivos externos que deben realizar la operación pedida.

Por ejemplo, como ya se ha hecho referencia, una de estas señales es la activación de la línea READ, que se actualiza cuando es necesario leer en la memoria un código de instrucción o bien un dato.

Análogamente, en el caso de escritura en memoria, para habilitar la operación se envía la señal WRITE.

El conjunto de las señales de control que la CPU envía o recibe del exterior puede dividirse, según sus funciones, en los siguientes grupos (ver gráfico de la página siguiente):

- señales para la transferencia de datos
- señales para el control del bus
- señales de sistema
- señales de sincronismo
- señales de interrupción.

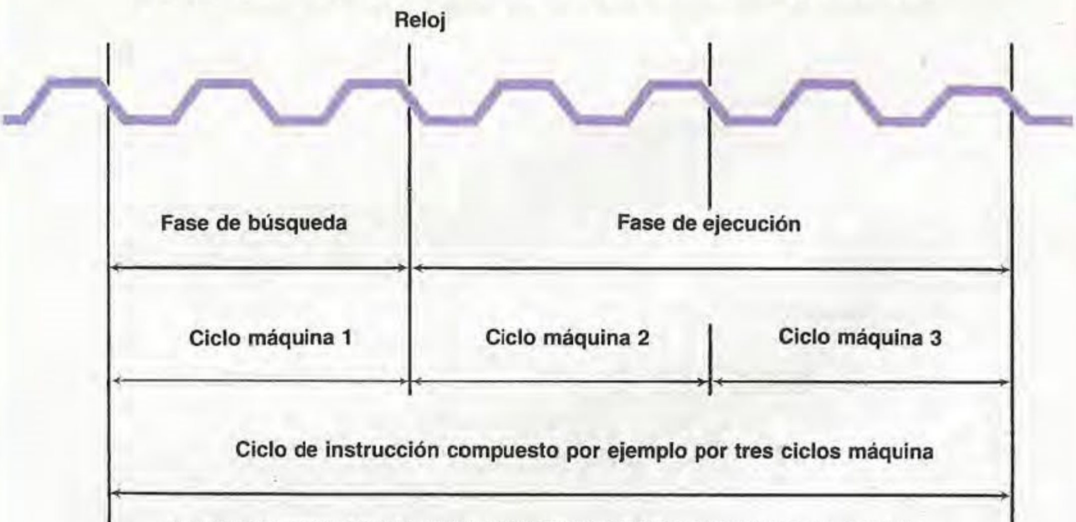
Al primer grupo pertenecen las señales que especifican si la operación de lectura (READ) o de escritura (WRITE) debe efectuarse hacia la memoria (MEMRQ) o hacia un dispositivo general de Entrada/Salida (IORQ). Las señales para el control del bus se utilizan cuando un dispositivo externo pide una transferencia de datos de o hacia la memoria, sin la intervención del microprocesador (operaciones de DMA, Direct Memory Access).

Generalmente, el intercambio en DMA se produce en tres fases. En primer lugar, el dispositivo envía una petición (señal BUSRQ, bus request) para tener asignado el bus (direcciones, datos y controles).

El microprocesador responde con la señal BUS AK (bus acknowledge, petición atendida) y suelta el control de los bus poniéndose en un estado de espera. Así, el dispositivo que ha pedido el DMA puede controlar el sistema sin interesar la CPU.

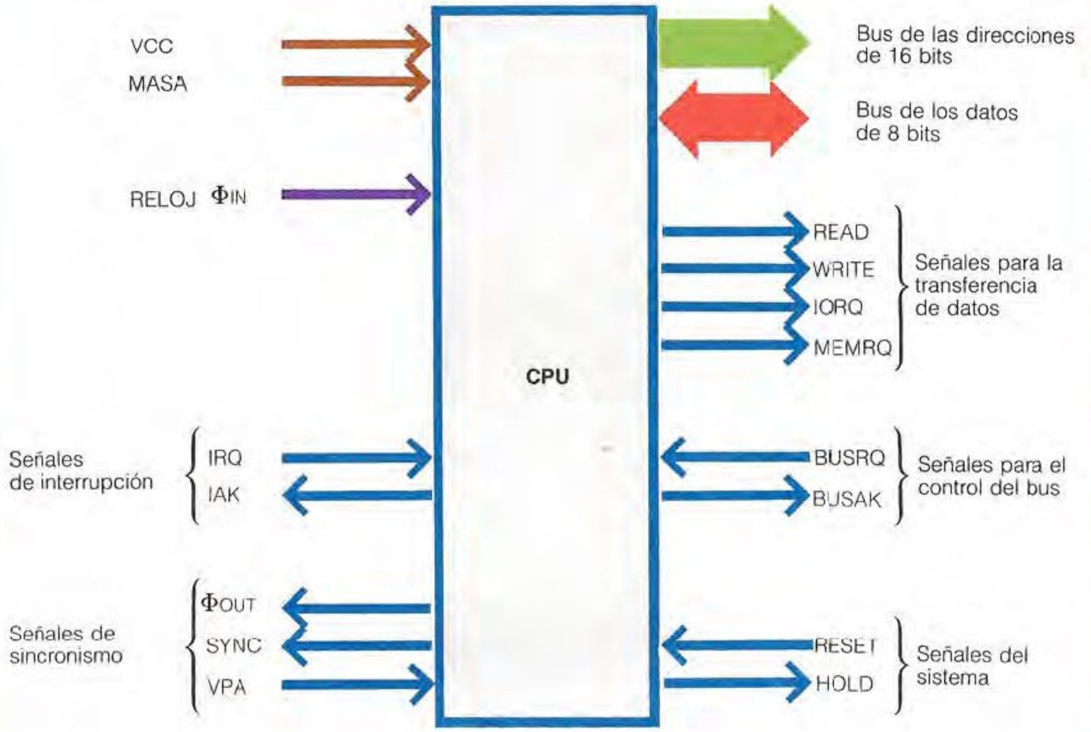
Durante el DMA, los dispositivos externos al microprocesador son informados del estado de inactividad de la CPU por la señal de HOLD, que junto con el RESET pertenece al grupo de las señales de sistema. La activación de la señal de RESET interrumpe cualquier actividad realizada por la CPU y coloca el sistema en las mismas condiciones anteriores al instante de la puesta en marcha.

DISTRIBUCION DE LOS CICLOS EN LA SEÑAL DE RELOJ



La fase de ejecución de una instrucción se divide necesariamente en dos partes principales: la fase de búsqueda y la fase de ejecución. Durante la búsqueda (un ciclo máquina) se toma el código de la instrucción, y durante la ejecución (que ocupa el número de ciclos máquina dependiente del tipo de instrucción) se activan todas las señales necesarias (bus de datos, direcciones, controles) para la ejecución efectiva de la instrucción

DESCRIPCION DE LAS LINEAS DE CONTROL DE UN MICROPROCESADOR



Algunos dispositivos conectados a la CPU necesitan, para su funcionamiento, recibir la señal de reloj del microprocesador, que pertenece al grupo de las señales de sincronismo. Esta señal (Φ_{out}, clock out) no es suficiente para sincronizar el dispositivo externo con la CPU, y son necesarias otras dos señales: la línea VPA (Valid Peripheral Address), que avisa al microprocesador cuando el dispositivo está preparado para iniciar la transferencia (lectura o escritura según las otras líneas de control) y la señal SYNC, que habilita o deshabilita la transferencia. El último grupo de las señales de control son las interrupciones, de las que hablaremos en breve. Por ahora diremos que las señales de interrupción previstas en los microprocesadores utilizados en los ordenadores personales necesitan líneas de control que, adecuadamente comandadas, activan las funciones de interrupción.

Finalmente, es necesario suministrar al microprocesador una alimentación, generalmente +5 voltios (VCC), y una masa (GROUND), y esto completa el examen de las líneas que van a parar a las patillas de un microprocesador.

La gestión de interrupción (interrupt)

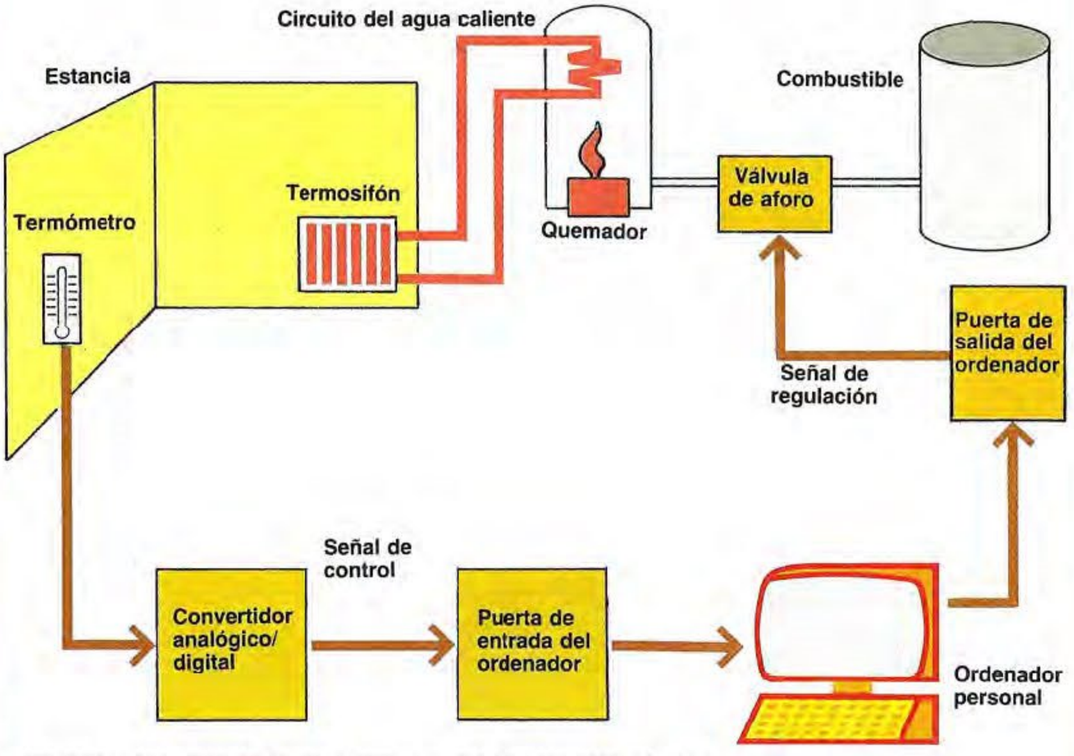
La función de interrupción consiste en la suspensión provisional del programa en ejecución en la CPU para dar la oportunidad a la propia CPU a responder a una petición procedente de un dispositivo externo, como puede ser, por ejemplo, una impresora o una unidad de disco. Una vez satisfecha la petición (mediante un salto o un subprograma de gestión del dispositivo externo), la CPU vuelve a la ejecución del programa interrumpido anteriormente.

La función de interrupción ya se describió en las págs. 132 y 133 del primer volumen. En este punto utilizaremos un ejemplo para ilustrar su funcionamiento práctico, es decir, el que se produce a nivel de la CPU cada vez que son activadas las líneas de control de la interrupción.

Supongamos que se desea utilizar un ordenador personal para controlar y regular la temperatura de un local o de un apartamento. Tendremos necesidad de un termómetro que mida la temperatura del ambiente y de un dispositivo que transforme el valor de la medición de temperatura en un número binario comprensible pa-

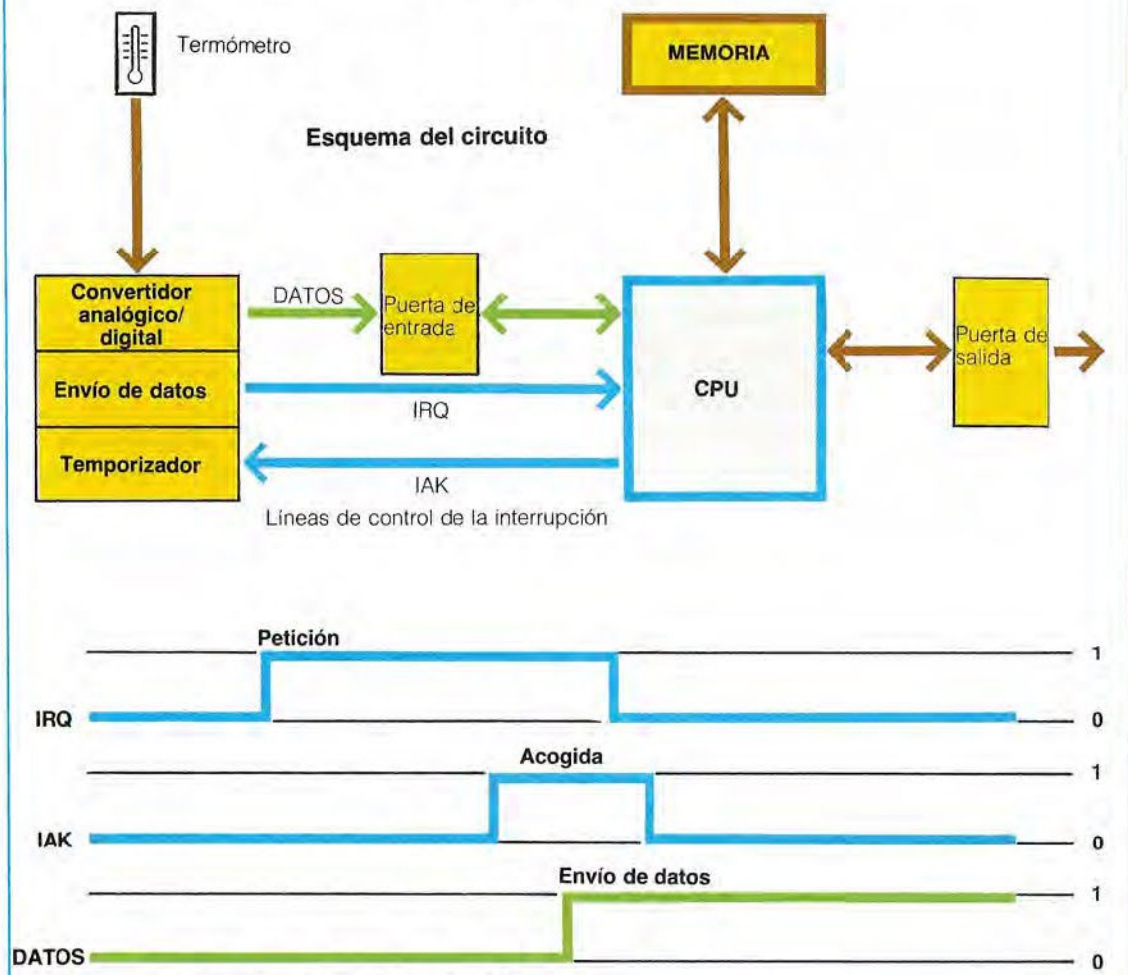
ra el ordenador. Después, el ordenador comparará este valor proporcional a la temperatura real, con la temperatura a la que queremos mantener el ambiente; en base al resultado de la comparación enviará los necesarios datos a un dispositivo de control, por ejemplo a la válvula de admisión de combustible en la caldera.
Abajo se muestra el esquema de realización de un control de este tipo. Para el funcionamiento del sistema puede pensarse que el ordenador comprueba de forma fija la temperatura presente en la puerta de entrada, la compara con el valor prefijado y envía continuamente las señales de regulación a la válvula de la caldera.
Este modo de funcionamiento no resulta el más adecuado, puesto que se basa en el aprovechamiento de la potencia del ordenador. Efectivamente, este último estaría empleado durante todo el tiempo en realizar operaciones banales, como la lectura de un dato de la puerta de entrada, la comparación de los números y la ejecución de un sencillo programa que transforma la diferencia entre dos números (el valor deseado y el valor real de la temperatura) en un dato a enviar a la puerta de salida. Por otra parte, los tiempos de reacción de los dispositivos térmicos son muy largos, es decir, que harían superfluo el control continuo. Es suficiente con medir la temperatura y hacer seguir la rutina de control al ordenador sólo a intervalos de tiempo precisados, por ejemplo cada minuto. En la parte restante del tiempo, el ordenador podría aprovecharse mejor para utilizarlo en otras tareas.
Con la interrupción es extremadamente fácil realizar un sistema de control que funcione según esta modalidad (ver gráfico de la derecha). Un temporizador conectado al convertidor analógico/digital genera, a intervalos prefijados de tiempo, una petición de interrupción poniendo el valor lógico 1 a la línea de control llamada IRQ (ver también el gráfico de la pág. 903). La CPU reconoce la petición de interrupción, interrumpe el programa que estaba realizando y queda disponible para responder a la petición poniendo

REGULACION DE TEMPERATURA REALIZADA POR MEDIO DE UN ORDENADOR



El converticor analógico/digital transforma la medida de temperatura en un número binario, que envía a la puerta de entrada del ordenador

FUNCIONAMIENTO CON INTERRUPCIONES DEL SISTEMA DE CONTROL DE TEMPERATURA



Proceso temporal de las señales

al valor lógico 1 la línea IAK. A la recepción de la señal IAK, el dispositivo externo envía a los dispositivos de entrada el dato que representa la medición de temperatura y anula la petición de interrupción poniendo a 0 la línea IRQ.

El ordenador envía entonces a ejecución el programa que se ocupa de leer el mensaje en la puerta de entrada, de comparar el valor de temperatura medido con el prefijado y de enviar la señal de control a la puerta de salida.

Después de haber completado la ejecución del programa de gestión de la interrupción, la CPU vuelve a realizar el programa que estaba en curso de desarrollo antes de la introducción.

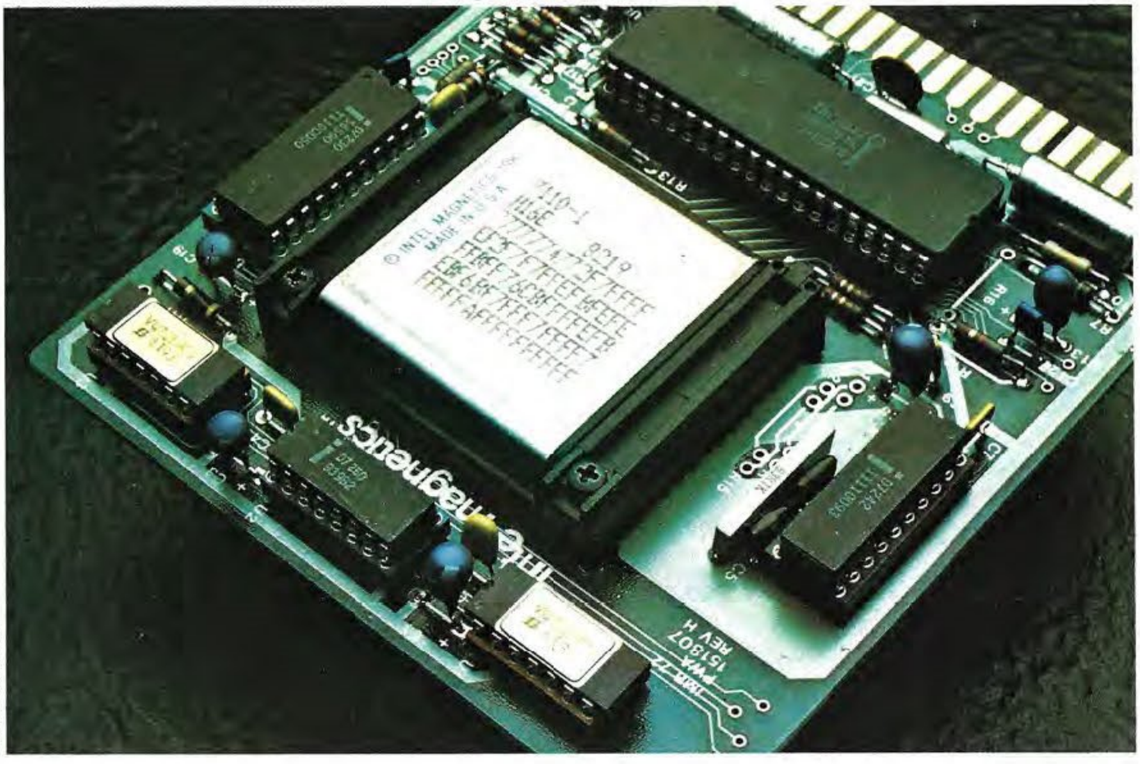
Analizaremos ahora el modo en que la CPU empieza a ejecutar la rutina de control de la temperatura y cómo hace después para reemprender la ejecución del programa interrumpido.

La pila (stack). Cuando la petición de interrupción se acepta, el dispositivo externo pone en el contador de programa la dirección inicial de la rutina de control; así, en el siguiente ciclo de fetch (instrucción de búsqueda) la CPU toma la primera instrucción de esta rutina, y no la prevista por el programa que estaba en ejecución. Sin embargo, la CPU, antes de aceptar la petición de interrupción, realiza una serie de operaciones necesarias para poder reemprender el programa que estaba realizando.

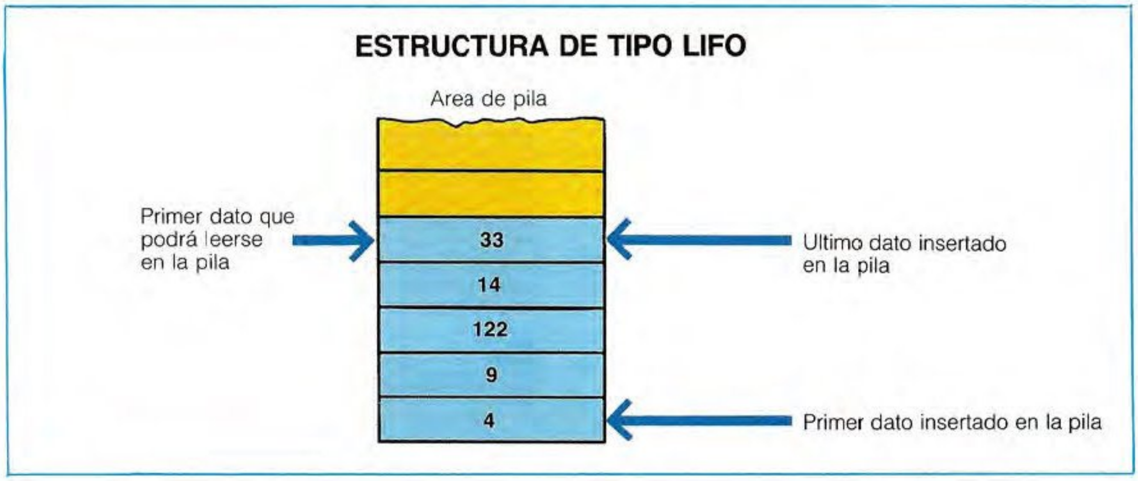
Efectivamente, el programa podría haber producido la memorización de informaciones im-

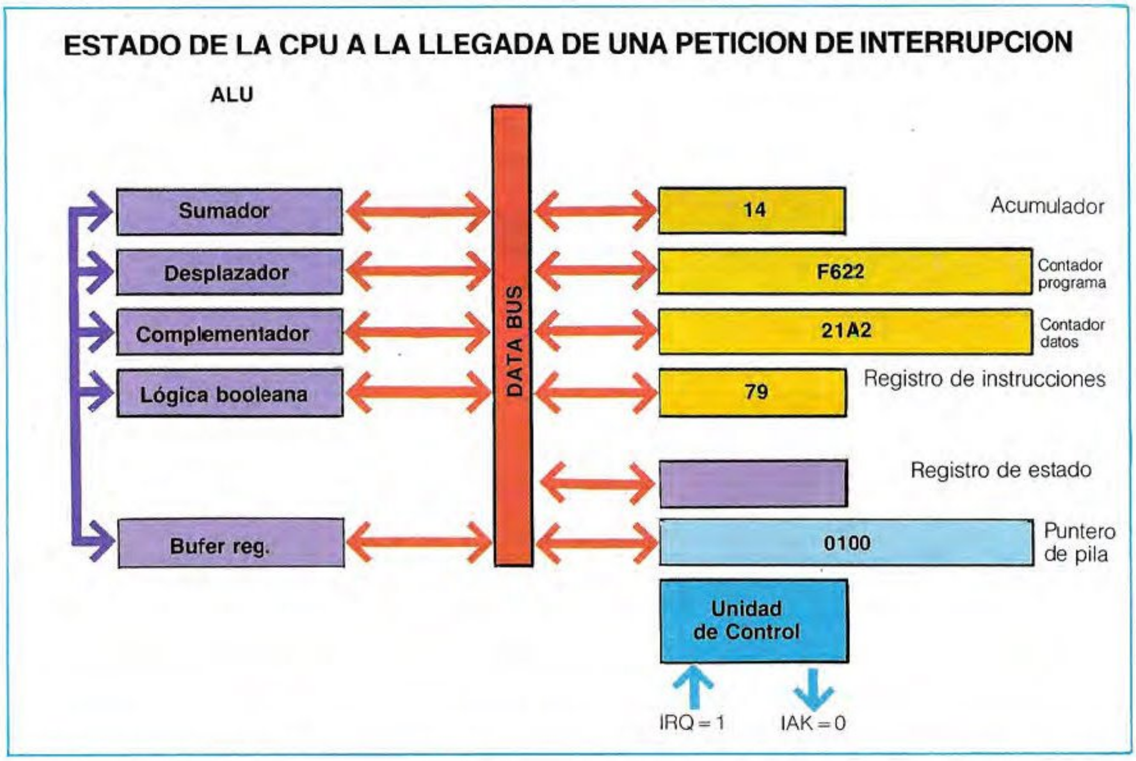
portantes en el registro de estado, en el contador de programa y en el acumulador; estas informaciones seguramente se perderían en el momento de realizar la rutina de control de la temperatura si no se procediese a salvarlas. Por tanto, antes de aceptar la interrupción, la CPU salva en un área de memoria el contenido de todos los registros. Es como si se memorizara una «fotografía» de la situación que precede a la aceptación de la interrupción. Terminada la ejecución del programa de gestión de la interrupción, en los registros de la CPU se recuperan los valores salvados anteriormente, y en el contador de programa se carga la dirección de la instrucción que debía realizarse en el momento de la interrupción. Por tanto, el programa inicialmente en ejecución puede reemprender su funcionamiento como si no se hubiese interrumpido. El contenido del registro de instrucciones no se salva, puesto que normalmente la CPU termina la ejecución de la instrucción en curso antes de aceptar la petición de interrupción. En consecuencia, el código contenido en el IR corresponde a una instrucción ya realizada y, por tanto, no es necesario salvarlo.
El contenido de los registros se salva en un área de memoria RAM constituida por varias células consecutivas denominada **pila** (stack) y que se utiliza también para salvar las direcciones de retorno de las subrutinas. Para gestionar la pila se usa un registro particular, el **puntero de pila** (stack pointer), en el cual se memoriza la situación «de carga» de la pila.
La CPU ve esta particular zona de memoria como una estructura de tipo Last-In First-Out (LIFO), o sea, como una pila de elementos en el que el último en llegar se entrega primero.
El funcionamiento de una pila del tipo LIFO se muestra en la página siguiente, arriba. El valor 33 se ha insertado en la pila el último y es el primer dato que puede leerse; el valor 14 podrá leerse sólo después del número 33. La pila funciona en la práctica como una serie de platos apilados uno encima de otro. Los platos podrán tomarse de la pila sólo en orden inverso a aquel en el que se han apilado.
En la página siguiente, en el centro, se muestra el esquema de la CPU a la llegada de una petición de interrupción. En el puntero de pila hay la dirección hexadecimal 0100, que apunta al principio del área de pila. La CPU salva el contenido de los registros a partir de esta dirección y, por

Una memoria de burbujas magnéticas de la firma Intel.



B. Coleman/Marka





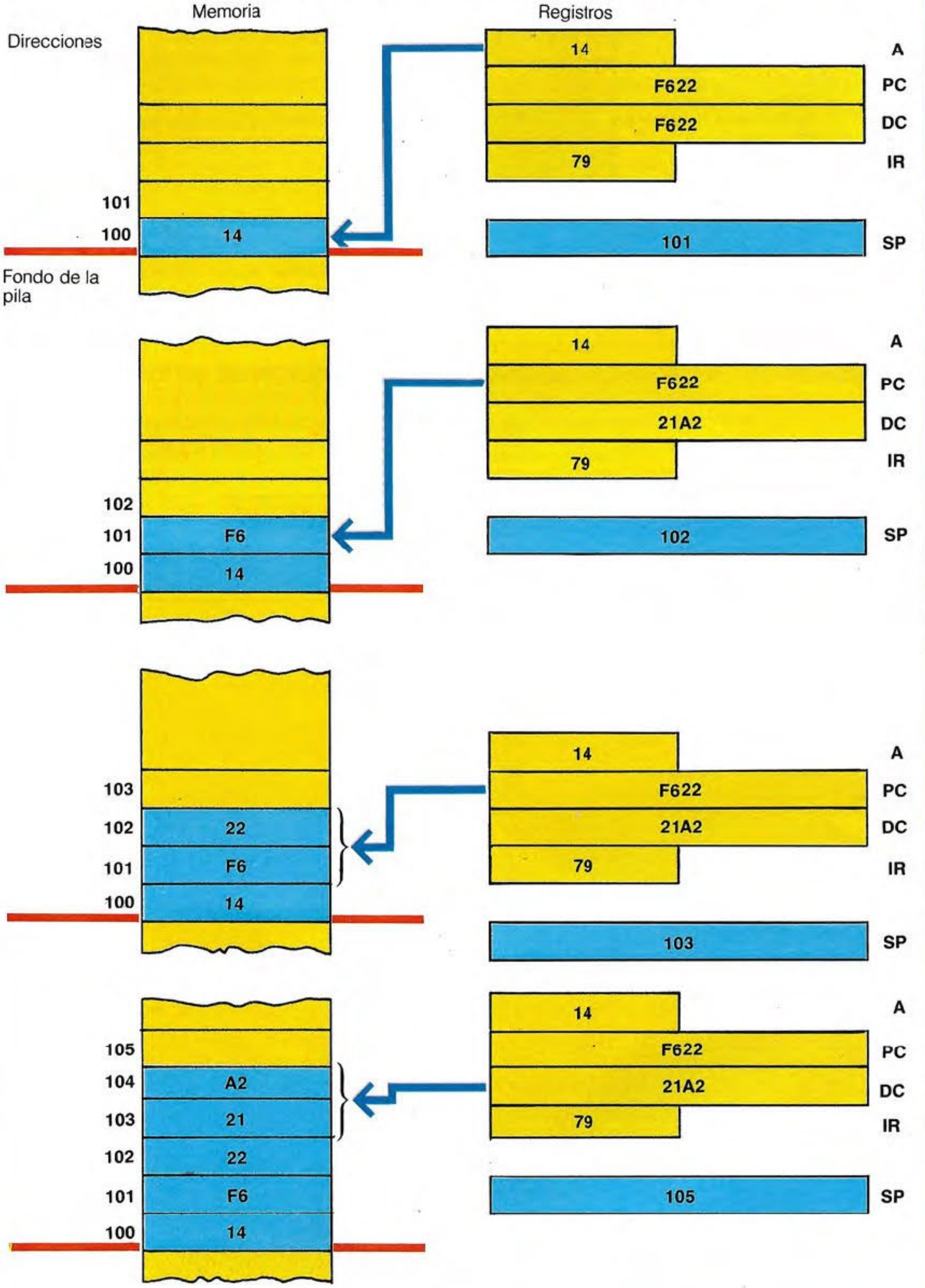
lo tanto, pone a 1 la línea IAK y acepta la interrupción solicitada.

La operación de salvamento se muestra detalladamente en la pág. 908. La CPU salva el contenido del primer registro en la pila escribiendo el dato en la celda de memoria direccionada por el puntero de pila, e incrementa después en 1 el valor de dicho puntero de pila. Los registros de dos bytes (PC y DC) están salvados, un byte en cada ocasión, en dos celdas de memoria contiguas. Para reactivar el programa suspendido se sigue la lógica inversa (ver pág. 909).

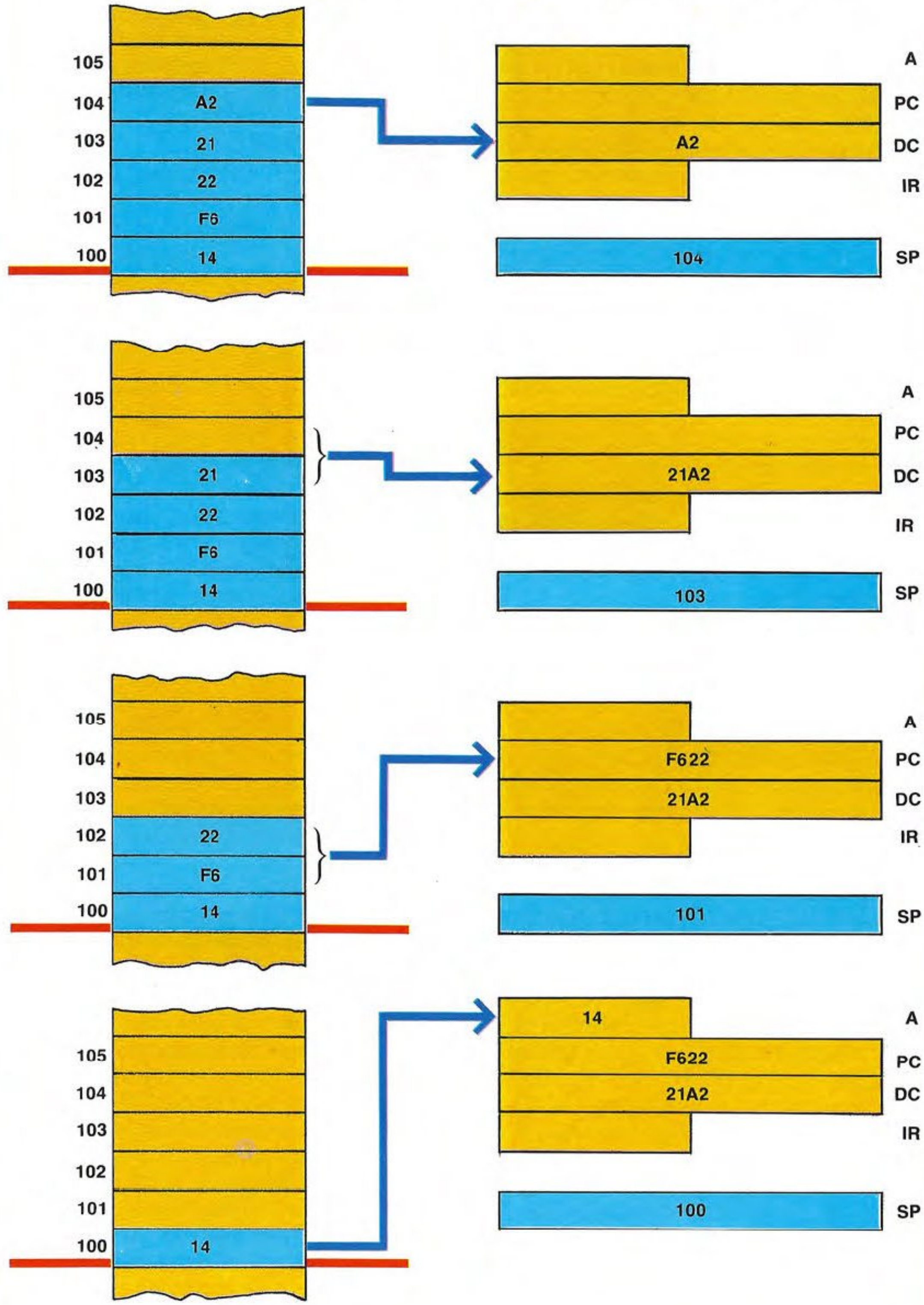
La operación de escritura en el área de pila puede continuar hasta el llenado total de dicha área, que en algunos sistemas tiene dimensiones fijas, mientras que en otros sistemas tiene una extensión que puede variar según las exigencias del programador.

En el lenguaje Assembler, las operaciones de escritura y lectura en el área de pila pueden realizarse fácilmente, puesto que existen instrucciones que actúan sobre el acceso del área incrementando o decrementando automáticamente el puntero de pila.

SALVAMENTO DEL CONTENIDO DE LOS REGISTROS EN EL AREA DE PILA A CONTINUACION DE LA INTERRUPCION



RECUPERACION DE LOS REGISTROS AL FINAL DE LA INTERRUPCION Y ANTES DE REACTIVAR EL PROGRAMA SUSPENDIDO



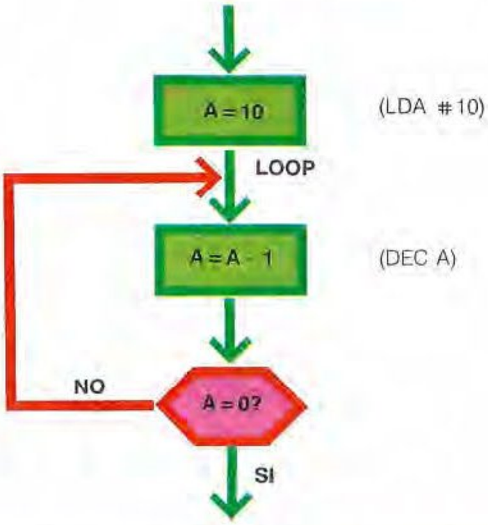
Estructura de las instrucciones Assembler

La CPU de un sistema de microprocesador tiene acceso a una memoria en la que hay registrados los códigos hexadecimales de las instrucciones de gestión de los registros, necesarios para la realización de las funciones previstas. A cada código-instrucción corresponde una secuencia de operaciones elementales (ciclos de máquina) que la CPU debe activar a la llegada de aquel código.
Normalmente, las instrucciones que componen el programa no se introducen en el microprocesador utilizando los códigos hexadecimales. Cuando se desea utilizar una simbología que sea lo más cercana posible al lenguaje máquina se emplea casi siempre el lenguaje Assembler.
Este lenguaje permite proporcionar las instrucciones utilizando un juego de códigos mnemónicos (no números hexadecimales) más fáciles de tratar, cada uno de los cuales encuentra correspondencia en un preciso código-instrucción hexadecimal. Sin embargo, se dice que **el lenguaje Assembler y el lenguaje máquina se corresponden uno a uno**.
La traducción de los códigos mnemónicos en los correspondientes códigos hexadecimales, directamente comprensibles por la CPU, se realiza por un programa de sistema llamado **Ensamblador**. El Ensamblador traduce una cadena de caracteres ASCII (código mnemónico, sin significado para la máquina) en un conjunto de bits que tienen un significado preciso para el microprocesador. En el interior de un ciclo de instrucción, en el primer ciclo máquina (ciclo de búsqueda) este conjunto de bits se carga en el registro de instrucción y es descodificado por la unidad de control; en base a las señales de control generadas, se toman de la memoria y se procesan los demás bits de la instrucción.
En los primeros calculadores sólo había la posibilidad de cargar los programas directamente en lenguaje máquina a través de un teclado hexadecimal o binario.
Los registros (como por ejemplo el contador de programa) se cargaban bit por bit, y la memoria se llenaba byte por byte.
Este sistema comportaba notables problemas de documentación y de corrección de los errores, además de la imposibilidad de releer con una cierta facilidad el programa escrito.
Precisamente para obviar estos inconvenientes se introdujeron los Ensambladores, que permitían utilizar los códigos mnemónicos, constituyendo el arquetipo de los lenguajes simbólicos de alto nivel más orientados al usuario.
Escribir una instrucción en forma mnemónica es más fácil que escribirla en forma de secuencia binaria, en la que un error banal, como por ejemplo el desplazamiento de un bit, compromete todo el código.
Por ejemplo, suponiendo que la instrucción «carga el acumulador con el valor 10» en Assembler tenga la forma LDA # 10*, el código binario correspondiente podrá ser del tipo 0 0 1 1 1 0 1 0 0 0 0 0 1 0 1 0 y resulta tener una forma poco clara y fácilmente desfigurable.
Cada instrucción Assembler ocupa una línea del programa fuente y se traduce en una instrucción única en código binario, llamada **instrucción máquina**.
Cada línea de programa y, por tanto cada instrucción, puede considerarse dividida en cuatro partes o campos, cada uno de los cuales tiene su significado preciso ligado a su posición. Los campos que componen una instrucción Assembler son: la **etiqueta** (label), el **código mnemónico** de la instrucción, los **operandos** y el **comentario**.
El campo etiqueta es un indicador de línea, necesario si en el interior del programa debe saltarse a aquella instrucción. El código mnemónico de la instrucción representa la parte que se descodificará en la unidad de control.
En cambio, el campo operando puede indicar la dirección de memoria en que se encuentra el operando o directamente el valor de dicho operando.
El campo comentario se utiliza sólo a fines de documentación. En la página siguiente se ilustra un segmento de programa en Assembler que ilustra la sintaxis descrita.
La primera instrucción del programa Assembler carga (mnemónico LD; en inglés, load significa carga) el valor 10 en el acumulador. La segunda instrucción decrementa (mnemónico DEC) en 1 el contenido del acumulador, mientras que la tercera efectúa un salto a la etiqueta LOOP si el contenido de A no es 0 (mnemónico BNZ, Branch Not Zero). Cuando A resulte igual a 0 se realizará la siguiente instrucción.
Es importante observar el método que se adopta para direccionar una cierta instrucción. Si el

* El significado del símbolo # se explica más adelante.

EJEMPLO DE INSTRUCCIONES ASSEMBLER

Etiqueta	Código mnemónico	Operando	Comentario
	LDA	#10	; Carga en el acumulador el valor 10
LOOP	DEC	A	; Decrementa en 1 el contenido del acumulador
	BNZ	A, LOOP	; Salta a "LOOP" si A es diferente de 0



programa tuviera que escribirse en Basic, una posible forma podría ser la siguiente:

```
100 A = 10
110 A = A – 1
120 IF A <> 0 GOTO 110
..............................................
```

La línea 120 direcciona (bajo condición) a la línea 110, y es el equivalente de la BNZ A, LOOP, con la diferencia de que el equivalente Assembler de la línea 110 tiene el nombre LOOP. En otros términos, si el intérprete Basic pudiese reconocer las líneas indicadas con un nombre simbólico, el programa Basic tendría la forma:

```
       A = 10
LOOP   A = A – 1
       IF A <> 0 GOTO LOOP
```

identificando la instrucción A = A – 1 con la etiqueta (label) LOOP y no con el número de línea. El paralelo descrito sólo es cualitativo, puesto que entre los dos programas existe una diferencia sustancial. En la versión Assembler, el símbolo A no es una memoria cualquiera como en el Basic, sino un registro preciso del microprocesador (el acumulador).
Por tanto, en realidad, los dos programas realizan funciones completamente diferentes. El programa Basic efectúa el cálculo y el test de condición sobre una memoria general, que por casualidad se llama A; el programa Assembler no involucra ninguna memoria, y realiza los cálculos y el test sobre el acumulador.

Métodos de direccionamiento

Antes de proceder al análisis de las instrucciones Assembler debe describirse la forma en que este lenguaje puede acceder a los datos residentes en memoria.
A cada memoria utilizada en un programa Basic hay asociado un nombre simbólico, y es con este nombre que puede reclamarse su contenido. En Assembler, las cosas son más complejas. Para utilizar una memoria debe **direccionarse**, es decir, se debe proporcionar al programa su dirección real y no un nombre simbólico. Los

métodos de direccionamiento usados más corrientemente en los microprocesadores, y por tanto en los correspondientes lenguajes Assembler, son los siguientes:

1 / Direccionamiento inmediato
2 / Direccionamiento absoluto o directo
3 / Direccionamiento indirecto
4 / Direccionamiento indexado
5 / Direccionamiento relativo
6 / Direccionamiento de registro
7 / Direccionamiento de pila

En algunos microprocesadores, estos modos de direccionamiento pueden combinarse entre sí para aumentar posteriormente la potencialidad de las instrucciones.
Examinemos ahora cada uno de los diversos modos de direccionamiento, introduciendo en los ejemplos algunas instrucciones Assembler que después se explicarán con más detalle, y que se referirán a un microprocesador de 8 bits.

Direccionamiento inmediato. El direccionamiento inmediato se tiene cuando el operando está contenido en la misma instrucción. Por ejemplo, la instrucción*

ADD A, #100

suma el valor 100 al contenido del acumulador, indicado con el código A. El símbolo # se ha adoptado para indicar que el número 100 es directamente el operando, y no su dirección. Abajo se muestra el formato típico de una instrucción de direccionamiento inmediato, es decir, que utiliza directamente como operando un valor indicado explícitamente (que es 100).
La ejecución de una instrucción de este tipo es muy rápida, puesto que la CPU, una vez cargada la instrucción, no debe realizar accesos posteriores de memoria para tomar el operando.

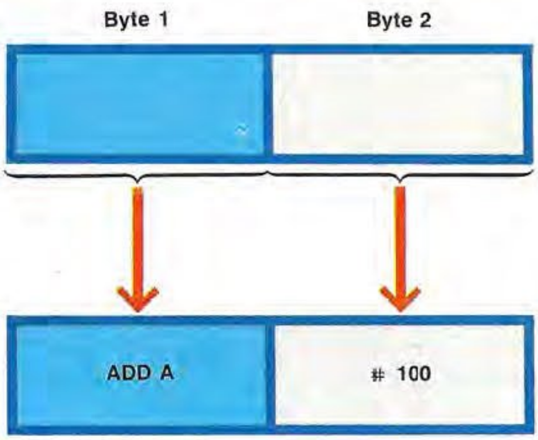
Direccionamiento absoluto o directo. Con este tipo de direccionamiento, la instrucción contiene la dirección de memoria del operando. Así

ADD A,/100

suma al acumulador el dato contenido en la dirección de memoria 100, depositando el resultado en el propio acumulador, como muestra el gráfico de arriba de la página siguiente.
El símbolo / se usa para identificar el direccionamiento directo. En el gráfico de arriba de la pág. 913 se muestra el formato típico de las instrucciones que utilizan el direccionamiento directo. Normalmente, estas instrucciones necesitan tres bytes, puesto que, además del código operativo (8 bits = 1 byte), son necesarios 16 bits para especificar una dirección entre las 64 k disponibles en memoria. Antes de tomar de la memoria el dato, la CPU debe efectuar otras dos

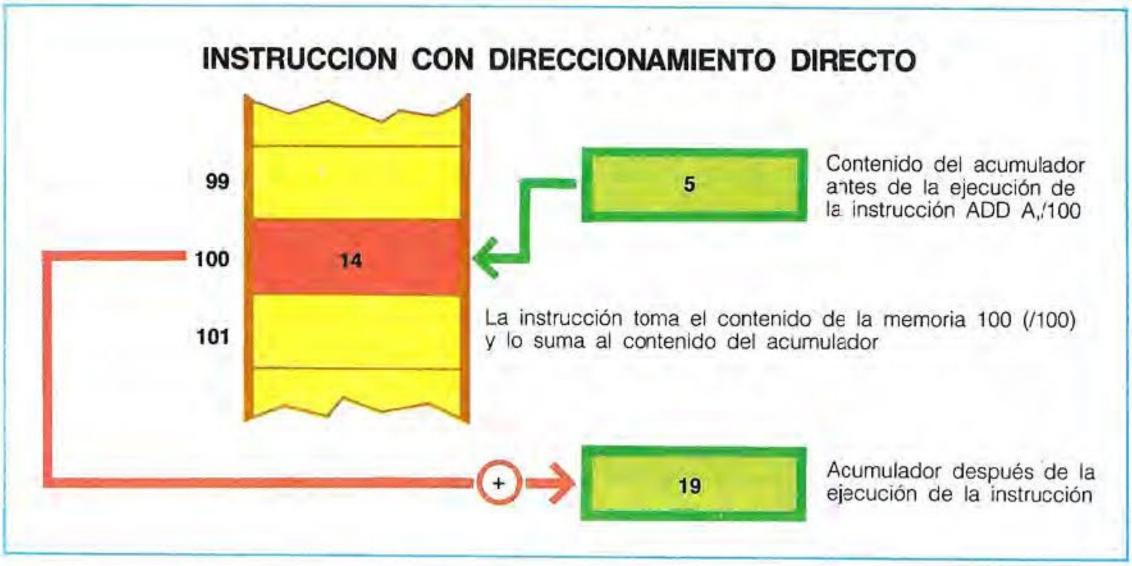
* Los códigos mnemónicos de los ejemplos tienen carácter indicativo. Para emplearlos realmente deben reescribirse en la forma prevista para el tipo de microprocesador particular.

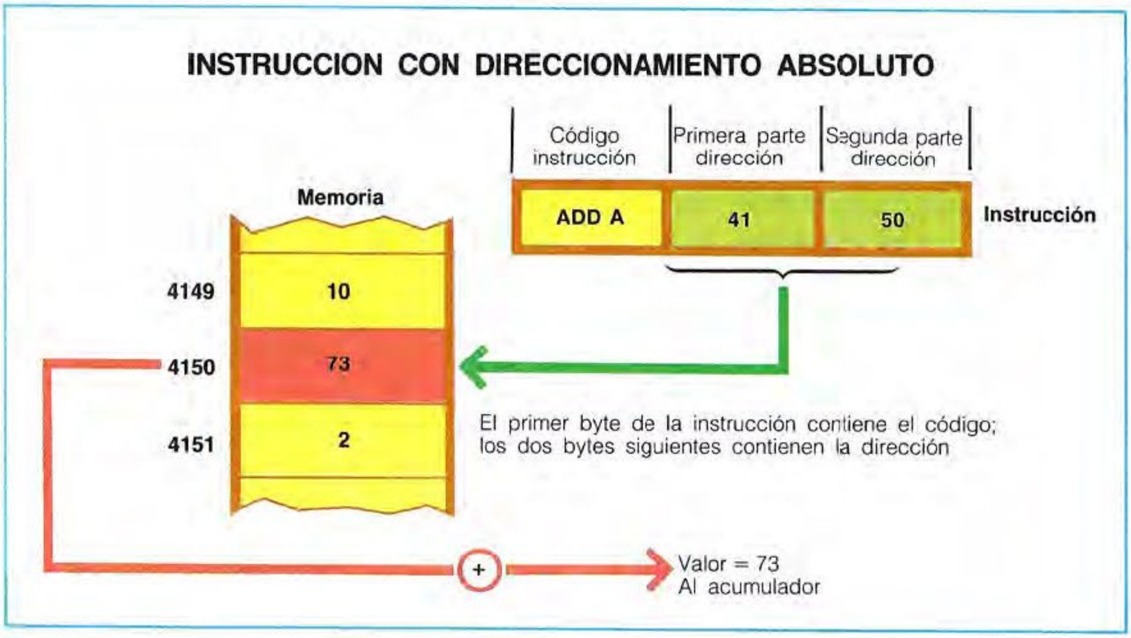
INSTRUCCION CON DIRECCIONAMIENTO INMEDIATO



La instrucción está compuesta por dos bytes. En uno hay escrito el tipo de operación a realizar (código), en el otro el operando

Al acumulador (A) se suma el valor 100; el resultado está en A



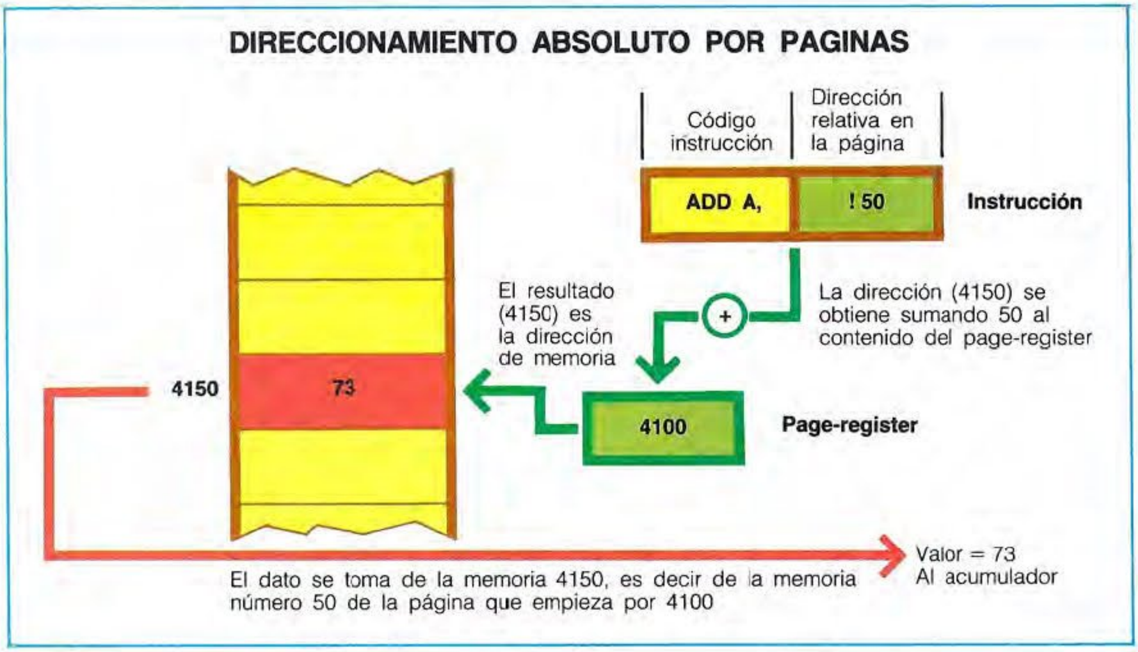


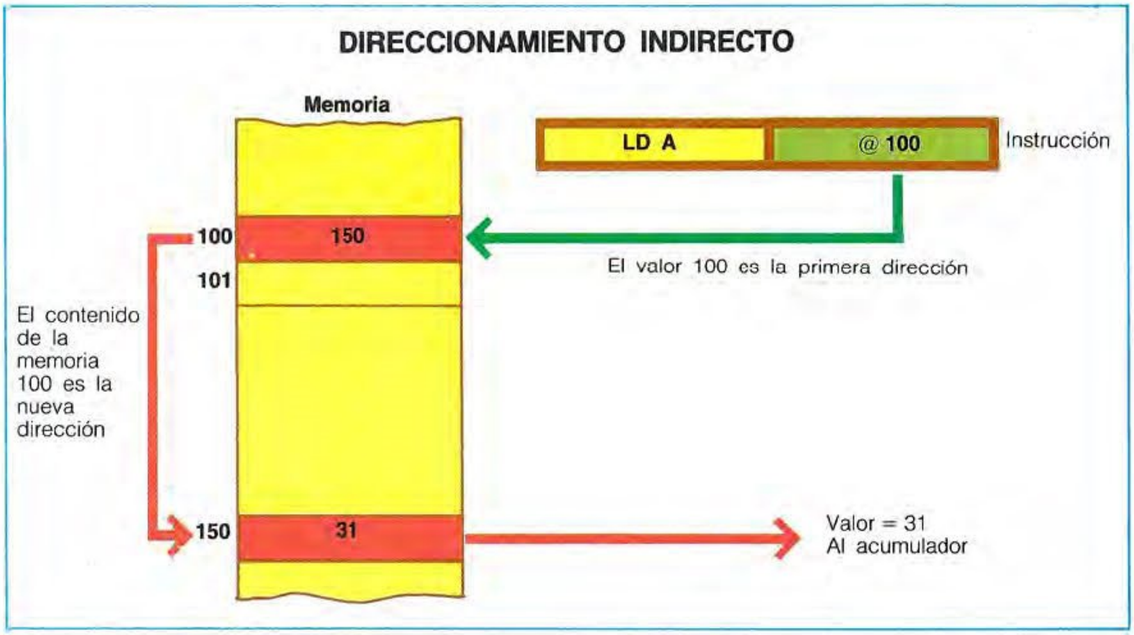
accesos para tomar los dos bytes de la dirección. Este tipo de direccionamiento puede hacerse más rápido subdividiendo la memoria en **páginas** de 256 bytes.

La instrucción direcciona uno de los 256 bytes de una página, mientras que un registro llamado **Page Register** contiene la dirección de partida de la página, que sumada a la dirección especificada por la instrucción determina la efectiva del operando (pág. 914, arriba). En el ejemplo, la instrucción es de dos bytes. Evidentemente es necesario que una instrucción anterior cargue el Page Register con el valor apropiado. Por tanto, este tipo de direccionamiento resulta conveniente cuando es necesario acceder a diversos datos contenidos en la misma página de memoria y, en consecuencia, no debe modificarse el contenido del Page Register.

Direccionamiento indirecto. En este direccionamiento, la celda de memoria direccionada por la instrucción no contiene el operando, sino la dirección de una memoria que lo contiene. Considerando por ejemplo la instrucción

LD A, @ 100





ésta cargará en el acumulador no el contenido de la memoria de dirección 100, sino el contenido de la celda de memoria cuya dirección está contenida en la celda 100, como se muestra arriba. La celda 100 contiene la dirección 150; el dato cargado en el acumulador será 31, o sea el contenido de la celda 150. En el ejemplo se ha adoptado el símbolo @ para especificar el modo de direccionamiento indirecto.

El direccionamiento indirecto comporta un funcionamiento más lento que el directo, puesto que la CPU debe efectuar dos accesos a la memoria antes de tomar el dato. Sin embargo, este tipo de direccionamiento resulta muy conveniente cuando se escriben subprogramas que procesan datos contenidos en áreas de memorias diferentes. El formato típico de la instrucción con direccionamiento indirecto es el mismo que para el direccionamiento directo; la longitud de la instrucción siempre es de 3 bytes.

Direccionamiento indexado. La CPU suma en este caso una dirección contenida en la instrucción con el contenido de un registro particular, llamado **registro índice**, para formar la dirección efectiva del operando. Indicando con X el contenido del registro índice, la instrucción

LD A,100(X)

hace que la CPU cargue en el acumulador el dato contenido en la celda cuya dirección es 100 + X (ver el gráfico de esta página).

Los paréntesis indican el modo de direccionamiento indexado.

A diferencia del Page Register, el registro índice no debe contener necesariamente la dirección inicial de una página en memoria, sino una dirección cualquiera.

El direccionamiento indexado se muestra muy potente para la elaboración de datos organizados en memoria en forma de vectores o matrices (array), y opera como se muestra en el gráfico de la pág. 916. Es más lento que el directo, porque la CPU debe sumar el contenido del registro índice a la dirección especificada en la instrucción antes de acceder a la memoria para tomar el dato. No obstante esto, los programas que utilizan el direccionamiento indexado resultan más rápidos cuando debe accederse a datos memorizados en celdas de memoria que tienen direcciones consecutivas.

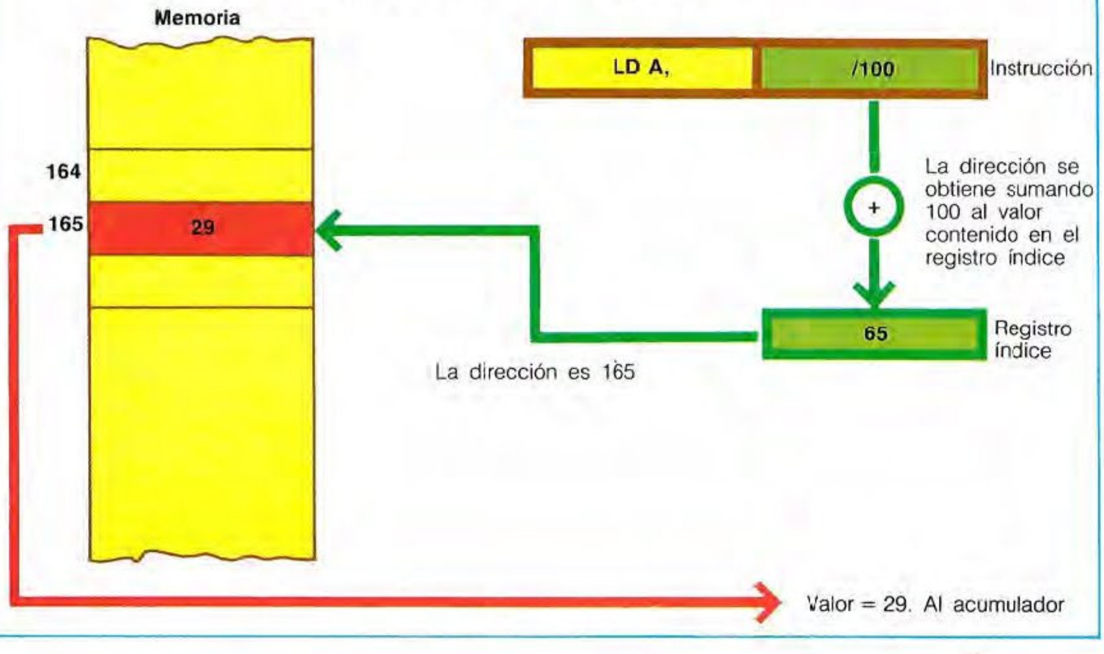
Direccionamiento relativo. Con este método, el direccionamiento del operando se determina sumando a la dirección contenida en la instrucción el valor contenido en el Contador de Programa.

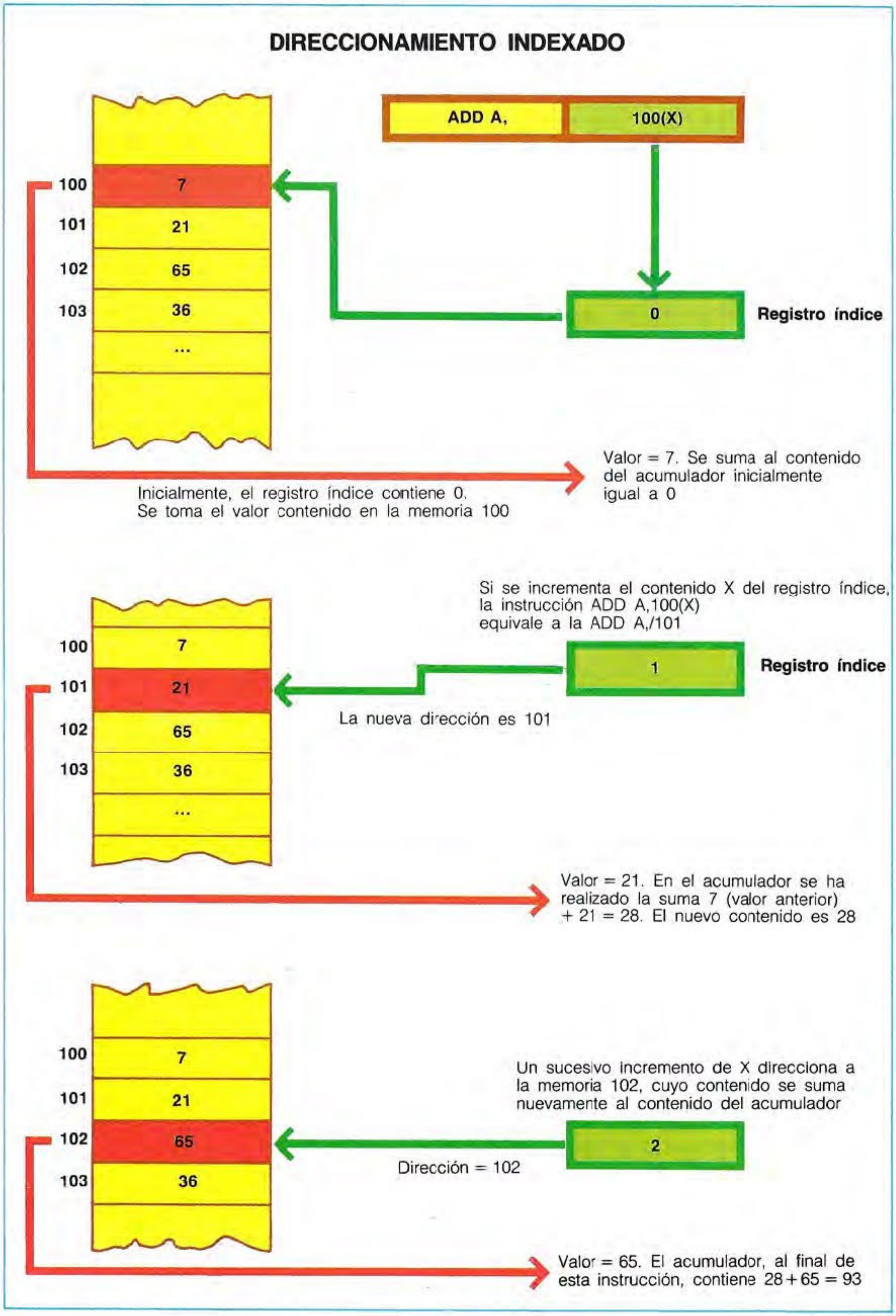
Indicando con el símbolo $ el direccionamiento relativo, y suponiendo que la instrucción en ejecución esté memorizada en la dirección 50, escribir LD A,$100 equivale a escribir LD A,/150, como muestra el gráfico de la pág. 917.

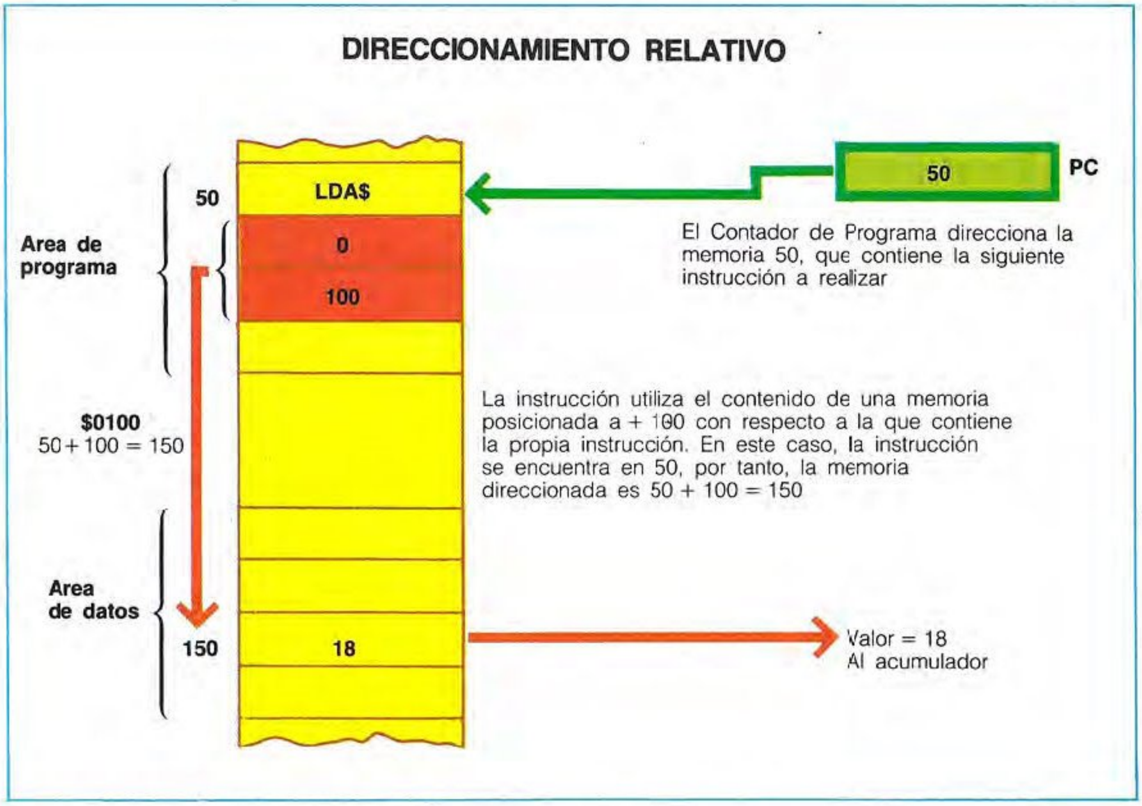
El direccionamiento relativo se utiliza cuando se crean programas reubicables, que pueden memorizarse de vez en cuando en áreas de memorias diferentes. Especificando las direcciones de los operandos de modo relativo con respecto a la dirección de la instrucción que la utiliza, al variar esta última no deben modificarse las direcciones de los operandos.

En otras palabras, el mecanismo consiste en direccionar una memoria identificándola con su posición relativa con respecto a una determinada dirección. De esta manera, el programa puede cargarse en una zona cualquiera de la memoria. Definiendo de vez en cuando el valor de la referencia (**base de reubicación**), todo queda inalterado.

DIRECCIONAMIENTO INDEXADO




DIRECCIONAMIENTO INDEXADO
ADD A,
100(X)
100
101
102
103
7
21
65
36
...
0
Registro índice
Valor = 7. Se suma al contenido del acumulador inicialmente igual a 0
Inicialmente, el registro índice contiene 0.
Se toma el valor contenido en la memoria 100
Si se incrementa el contenido X del registro índice, la instrucción ADD A,100(X) equivale a la ADD A,/101
1
Registro índice
La nueva dirección es 101
Valor = 21. En el acumulador se ha realizado la suma 7 (valor anterior) + 21 = 28. El nuevo contenido es 28
Un sucesivo incremento de X direcciona a la memoria 102, cuyo contenido se suma nuevamente al contenido del acumulador
2
Dirección = 102
Valor = 65. El acumulador, al final de esta instrucción, contiene 28 + 65 = 93



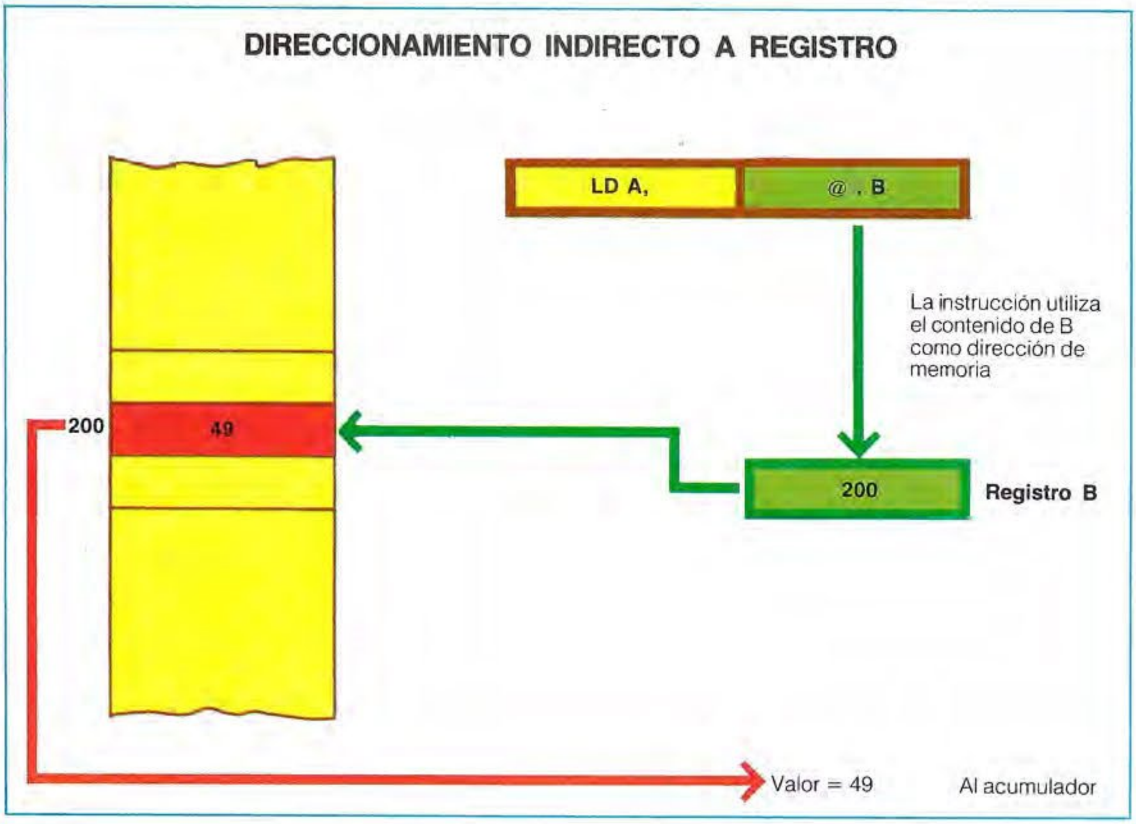
Direccionamiento de registro. Se tiene cuando en la instrucción se especifica la dirección de un registro en lugar de una dirección de memoria. Los registros están contenidos en la CPU y tienen un número limitado. Este método de direccionamiento permite hacer mucho más rápida la ejecución de las instrucciones y reducir la longitud del código. Por ejemplo, la instrucción

ADD A,.B

suma al contenido del acumulador el contenido del registro B. El símbolo utilizado para distinguir este tipo de direccionamiento es el punto. La ejecución de una instrucción de este tipo es muy rápida, puesto que no son necesarios accesos a la memoria.
Evidentemente es necesaria una instrucción anterior que cargue en el registro B el valor que se sumará después en A.
El direccionamiento de registro resulta muy útil cuando debe utilizarse muchas veces un valor constante en un proceso. Puede utilizarse también de forma indirecta, como se ve en la pág. 918, utilizando el contenido del registro (B) como dirección del valor a tomar. También en este caso, la ejecución de la instrucción resulta más rápida, puesto que se efectúa un solo acceso a memoria.

Direccionamiento de pila. Este tipo de direccionamiento prevé que los datos utilizados por una cierta instrucción se memoricen en la pila. El direccionamiento de pila tiene notables ventajas, puesto que las instrucciones son más cortas y, por tanto, tienen un tiempo de ejecución muy breve. Efectivamente, después de cada acceso a la pila, la CPU recalcula automáticamente la dirección de entrada en la pila, por lo que el programador no debe prever instrucciones para la actualización de las direcciones. El direccionamiento de pila también puede utilizarse en modo directo e indirecto.

Generalmente, no todos los métodos de direccionamiento pueden utilizarse con todos los tipos de instrucción. Sólo algunas formas particulares de código binario, llamadas **códigos ortogonales**, prevén esta posibilidad. En la realidad, los códigos de este tipo se emplean raramente, puesto que los fabricantes prefieren especializar las instrucciones.



Las instrucciones Assembler

Ya se ha dicho varias veces que las instrucciones Assembler permiten comunicar a la máquina comandos e instrucciones relacionados uno a uno con los códigos hexadecimales propios del lenguaje máquina,. pero utilizando códigos mnemónicos alfabéticos y numéricos.

Es muy importante comprender que el código binario de una determinada instrucción es único y, normalmente, inalterable por parte del usuario, puesto que está escrito en el interior de la CPU por el fabricante, de manera indeleble e idéntica para todos los microprocesadores del mismo tipo. En cambio, la codificación mnemónica Assembler es modificable en cualquier momento; basta con disponer de un Ensamblador diferente. Este último podrá aceptar un código mnemónico también completamente diferente al anterior, pero cada instrucción se traducirá siempre y únicamente a los códigos binarios previstos por aquel microprocesador.

Técnicamente no existe ninguna razón por la que cada tipo de microprocesador utilice conjuntos de códigos mnemónicos diferentes entre sí, pero actualmente cada fabricante usa simbologías propias, particulares de cada microprocesador. La ausencia de una norma entre los diversos Assembler disponibles conlleva notables problemas en la fase de escritura de los programas; a menudo es necesario reescribir completamente rutinas ya desarrolladas sólo porque cambia el microprocesador utilizado. Motivado por esta dificultad, el Instituto de Ingeniería Eléctrica y Electrónica (IEEE, Institute of Electrical and Electronics Engineering, Task T 694/D) ha propuesto una norma a utilizar en la definición de las instrucciones.

El lenguaje Assembler propuesto por el IEEE, al cual nos referiremos después, quizá puede parecer más complejo de lo necesario, puesto que define reglas precisas a respetar, y en algunos casos es además redundante. Sin embargo, debe tenerse en cuenta que a este nivel, la excesiva compactación de un lenguaje puede hacerlo ilegible, mientras que si se fijan algunas reglas fundamentales para todas las instrucciones, lo que parecen complicaciones llevan en realidad a una mayor legibilidad del programa.

Por tanto, el programador debe conocer a fondo el microprocesador que está utilizando, porque no todas las instrucciones existentes están dis-

ponibles en una determinada CPU. En general, en un lenguaje Assembler están presentes los siguientes tipos de instrucciones:

1 / Instrucciones condicionales
2 / Instrucciones aritméticas
3 / Instrucciones lógicas
4 / Instrucciones de transferencia de datos
5 / Instrucciones de bifurcación (Branch)
6 / Instrucciones de salto de una instrucción (Skip)
7 / Instrucciones de llamadas a subrutinas
8 / Instrucciones de retorno a subrutinas
9 / Instrucciones varias

Todos los códigos mnemónicos están formulados respetando la regla fundamental que prevé la formación de cada palabra-instrucción con las iniciales de la palabra que compone la descripción de la acción activada. Por ejemplo,

ADDC = ADD with Carry = Suma teniendo en cuenta los acarreos
BGT = Branch if Greater Than = Salta si el contenido del acumulador es mayor que...

Cuando es necesario, en el código mnemónico también hay especificado el tipo de operando al que se refiere la instrucción.
El carácter de especificación es

B si el operando es un byte
H si el operando es una halfword (media palabra)
L si el operando es una longword o doubleword (doble palabra)
D si el operando es un decimal
F si el operando es una coma flotante
I si el operando es un bit
4 si el operando es un nibble (4 bits) o un dígito

A falta de una declaración explícita, el operando se supondrá por omisión que es de tipo word (o sea una palabra, 16 bits).

Instrucciones condicionales

Son aquellas instrucciones que permiten condicionar la ejecución del programa al producirse o no un determinado suceso.
Generalmente, la condición está definida por los bits del registro de estado. En otros términos, el suceso se ha representado con el valor verdadero o falso (0 o 1) de uno de los bits cero (Z), acarreo (C), desbordamiento (O o V), signo (N), paridad (P) o de una combinación de ellos.
Las instrucciones como las de salto (branch), de salto de una sola instrucción (skip) o de una llamada a un subprograma (call), pueden ser condicionadas, por ejemplo, con respecto a uno de los siguientes sucesos:

Z Cero: la instrucción se realiza si el bit de cero tiene un valor lógico 1 (condición verdadera), o sea si el resultado de la última operación realizada ha sido cero
NZ No Cero: la instrucción sólo se realiza si el bit cero tiene un valor lógico 0, o sea si el resultado de la última operación realizada no es cero

Hay otras instrucciones que pueden vincularse a operaciones relacionales de comparación. Este tipo de condicionamientos se impone siempre sobre la base de los valores de los bits del Registro de Estado a través de algunas operaciones lógicas.

Empleo de un lenguaje simbólico orientado al problema de la gestión de tráfico ferroviario.


Reese/Marka

TESTS CONDICIONALES

Condición	Código mnemónico
Zero	Z
Not Zero	NZ
Carry	C
Not Carry	NC
Negative	N
Positive	P
Overflow	V
Not Overflow	NV
Equal	E
Not Equal	NE
Greater Than	GT
Less Than	LT
Greater or Equal	GE
Less or Equal	LE
Higher	H
Lower	L
Not Higher	NH
Not Lower	NL

En la tabla de al lado se han indicado los códigos de algunos tests condicionales. Hay que recordar que las acciones dependen del microprocesador en uso, y sobre todo de la estructura del Registro de Estado.
También es muy importante recordar que no siempre todos los bits del Registro de Estado son modificados por una instrucción. En particular, en las instrucciones de salto (branch o skip) es indispensable saber qué bits han variado con posterioridad a la ejecución de la instrucción anterior.
Generalmente, en los manuales Assembler existen tablas que, instrucción por instrucción, dicen qué bits se modifican, cuáles permanecen inalterados y cuáles son indefinidos.
En la tabla de abajo se ha representado un ejemplo.

Instrucciones aritméticas

En la tabla de la página siguiente se indican algunas instrucciones aritméticas que normalmente están previstas por los Ensambladores.

EXAMEN DE LOS CODIGOS DE CONDICION (Tabla teórica)

Operación	Registro de Estado	Tipo de operación
	C Z O N H P	
ADD	* * * * * *	Suma
AND	0 * 0 * – *	AND lógico
LD	– – – – – –	Carga de registro
SHL	* * 0 * U *	Desplazamiento a la izquierda
ST	– – – – – –	Memorización

*** situado según el resultado de la operación**
– no modificado por la operación
0 puesto a cero en cualquier caso
U (undefined) indefinido después de la operación (es decir modificado, pero sin significado)

ALGUNAS INSTRUCCIONES ARITMETICAS

Instrucción	Código mnemónico	Descripción
Add (suma)	ADD	Realiza una suma
Add with carry (suma con acarreo)	ADDC	Suma con acarreo de anteriores operaciones
Subtract (resta)	SUB	Resta
Subtract with carry (resta con acarreo)	SUBC	Resta con acarreo procedente de una operación anterior
Increment (incremento)	INC	Suma 1 al operando
Decrement (decremento)	DEC	Resta 1 al operando
Multiply (multiplicación)	MUL	Multiplicación
Divide (división)	DIV	División
Compare (comparación)	CMP	Comparación entre dos operandos
Negate (negación)	NEG	Realiza un complemento a 2 del operando

Estas instrucciones permiten realizar todas las operaciones de suma sin (ADD) y con acarreo (ADDC).

Las operaciones con acarreo se utilizan, cuando es necesario tener una mayor precisión, para sumar números representados por más bytes.

Las instrucciones de incremento o decremento suman o restan un 1 al operando especificado. Por ejemplo, la instrucción INC A incrementa en 1 el contenido del acumulador. Estas instrucciones son muy útiles para la actualización rápida de los contadores.

Las instrucciones de multiplicación (MUL) y de división (DIV) están presentes sólo en los microprocesadores más sofisticados (generalmente los de 16 bits), puesto que las correspondientes operaciones no se realizan a través de circuitos específicos, sino mediante oportunas señales de control enviadas por la Unidad de Control al sumador y al complementador.

En la operación de comparación se realiza una comparación entre los dos operandos. Generalmente, esta operación se realiza mediante una resta entre los dos operandos; en base al examen del bit de cero resultará la igualdad o la desigualdad de los dos operandos.

La operación de complemento a 2 se efectúa a través del circuito de complemento, y es posible activarla con la instrucción NEG. Por ejemplo, introduciendo NEG A, el contenido del acumulador se complementa a 2.

Si el contenido del acumulador es $(4)_{10}$

MSB							LSB	
0	0	0	0	0	1	0	0	Acumulador

ejecutando la instrucción NEG A tendremos

	64	32	16	8	4	2	1	
1	1	1	1	1	1	0	0	Acumulador

es decir $(-124)_{10}$ teniendo en cuenta que el bit más significativo (MSB) es el del signo.

Instrucciones lógicas

Las principales instrucciones lógicas normalmente presentes en el lenguaje Assembler se relacionan en la tabla de las págs. 923 y 924.
La instrucción más usada es la AND, porque con ella es posible efectuar operaciones de enmascarado, es decir es posible seleccionar algunos bits de una palabra a través de una máscara para examinar a continuación su valor.
Por ejemplo, supongamos que quiere determinarse si en el contenido del acumulador el bit 5 vale 1 o 0. El método más rápido consiste en efectuar una operación de AND entre el acumulador y un determinado valor numérico, para evidenciar sólo el bit 5. El número a utilizar, es decir la máscara, debe ser tal que todos los bits del acumulador resulten siempre 0, excepto el bit 5, que debe conservar el valor real. En nuestro caso, la máscara deberá hacerse así

MSB			bit. n.				LSB
7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0

El resultado de la operación Acumulador AND Máscara es cero si el bit 5 vale cero, mientras es diferente de cero si el bit 5 del acumulador vale 1. En sustancia, se trata de la misma operación AND vista en el Basic. Un test posterior sobre el bit cero del Registro de Estado proporciona el valor (0/1) del bit aislado así.
En la figura de la pág. 925 se ilustra el diagrama de flujo y el programa Assembler que realiza la operación descrita.
La instrucción XOR puede utilizarse para poner a cero el acumulador, aprovechando el hecho de que la OR exclusiva de cualquier número con sí mismo es cero

XOR A,A

0 1 0 1 0 0 1 1	Valor (acumulador)
XOR	
0 1 0 1 0 0 1 1	Valor (acumulador)
0 0 0 0 0 0 0 0	Resultado (al acumulador)

La instrucción:

LOAD A, 0

(carga en el acumulador el valor inmediato cero) obtiene el mismo resultado (A = 0) pero con tiempos más largos.

Las instrucciones Shif y Rotate son útiles para convertir datos de la forma serie a la forma paralela, para normalizar o escalar números, para compactar datos antes de memorizarlos o para separarlos antes de utilizarlos, para aislar bits sencillos o partes de palabra, para realizar multiplicaciones y divisiones. La instrucción Test permite modificar los bits del Registro de Estado y realiza una resta entre el operando especificado y el valor cero. El resultado de la operación no se memoriza, aunque se varían los bits condicionales (Registro de Estado) de acuerdo con el resultado del test. De esta manera es posible cambiar los flags condicionales sin mover datos. La instrucción Test es análoga a la Compare (CMP), que en cambio efectúa una resta entre los dos operandos, de la que generalmente uno es el contenido del Acumulador y el otro es el contenido de una celda de memoria.
También la instrucción Compare no transfiere ningún resultado al acumulador, pero cambia los bits del Registro de Estado, que pueden utilizarse para condicionar saltos. Por ejemplo, la instrucción BZ LOOP (es decir salta si cero) realiza un test sobre el bit de cero del Registro de Estado y salta a la etiqueta LOOP si el bit de cero vale 1.

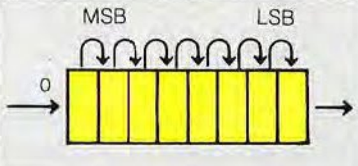
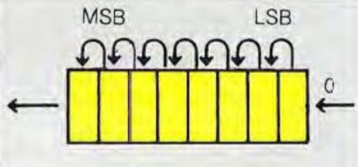
Instrucciones de transferencia de datos

En este grupo están comprendidas todas las instrucciones de intercambio de datos entre el microprocesador y sus periféricos.
Las transferencias tienen lugar principalmente entre la memoria y los registros del microprocesador, y las instrucciones utilizadas normalmente son Load y Store. La primera permite cargar el registro especificado (primer operando o destino) con el valor dado como segundo operando (fuente, origen).
Por ejemplo, la instrucción

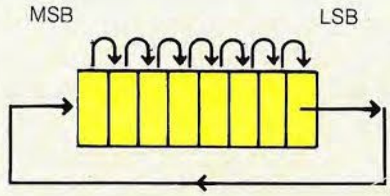
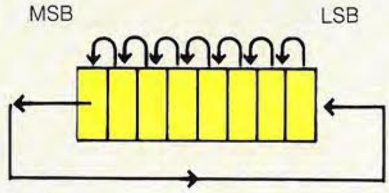
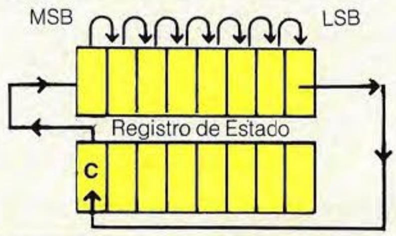
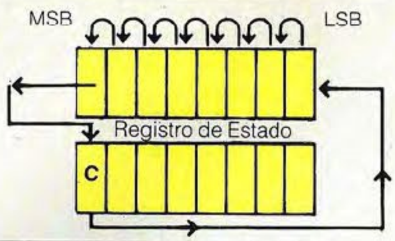
código instrucción	primer operando	segundo operando
LD	A,	0

tiene la misión de poner a cero el contenido del acumulador A (primer operando). El segundo operando indica directamente el valor a cargar (o sea sin acceder a la memoria).
La instrucción Store representa el aspecto doble con respecto a la Load. A través de ésta es posible transferir el contenido de un registro en una celda de memoria especificada con la di-

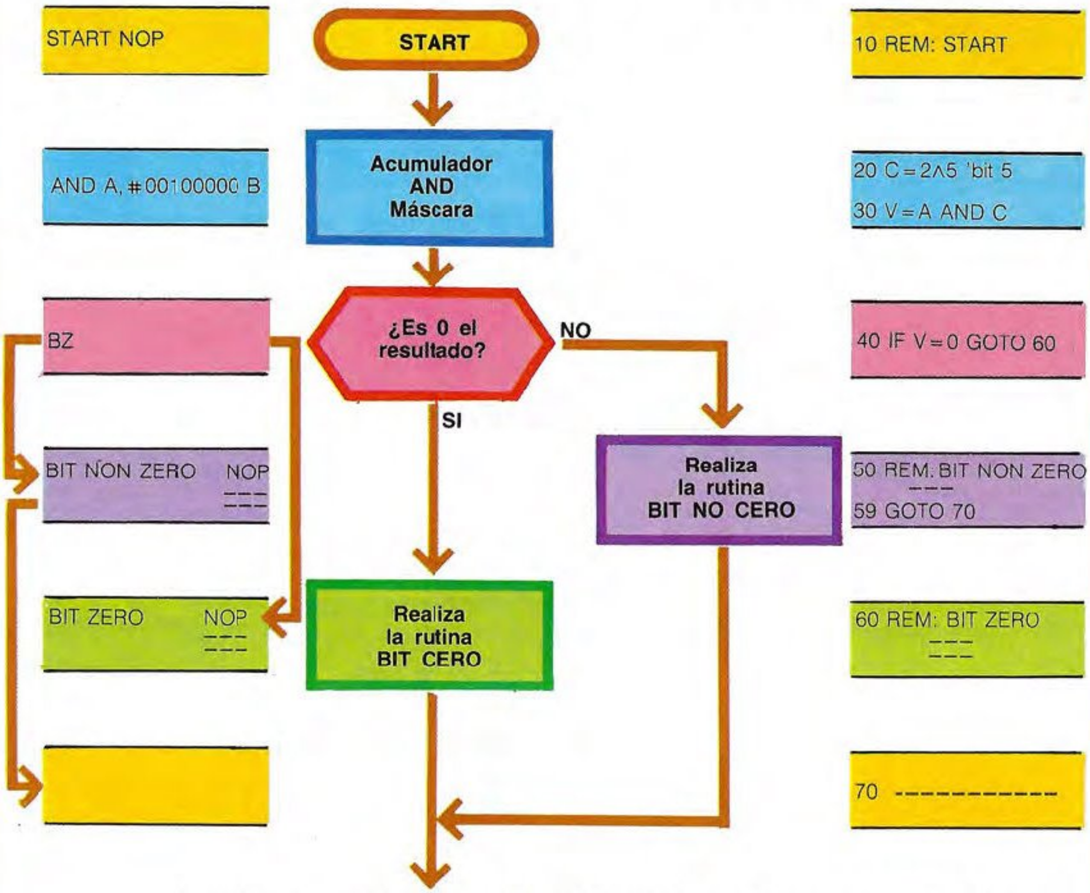
INSTRUCCIONES LOGICAS

Instrucción	Código mnemónico	Descripción	Ejemplo
And	AND	Operaciones de AND lógico	op1 0 1 0 1 0 0 1 1 op2 0 1 1 0 0 1 0 1 Res 0 1 0 0 0 0 0 1
Or	OR	Operaciones de OR lógico	op1 0 1 0 1 0 0 1 1 op2 0 1 1 0 0 1 0 1 Res 0 1 1 1 0 1 1 1
Exclusive or	XOR	Operaciones de OR exclusivo	op1 0 1 0 1 0 0 1 1 op2 0 1 1 0 0 1 0 1 Res 0 0 1 1 0 1 1 0
Not	NOT	Operación de complemento del operando	op1 0 1 0 1 0 0 1 1 Res 1 0 1 0 1 1 0 0
Not carry	NOTC	Complemento del bit de acarreo en el Registro de Estado	C Estado antes 1 Estado después 0
Shift right (desplazamiento a la derecha)	SHR	Operación de desplazamiento de uno o más puestos a la derecha (hacia el LSB) con la sustitución de los bits más significativos con ceros	
Shift left (desplazamiento a la izquierda)	SHL	Los bits del operando se desplazan uno o más puestos a la izquierda (hacia el MSB). Los bits menos significativos se ponen a cero	
Shift right aritmetic (desplazamiento aritmético a la derecha)	SHRA	Desplazamiento de los bits del operando uno o más puestos a la derecha (hacia el LSB). El bit más significativo (MSB), es decir el del signo, se mantiene en su posición, y se copia simplemente en el bit adyacente.	

INSTRUCCIONES LOGICAS

Instrucción	Código mnemónico	Descripción	Ejemplo
Rotate right (rotación a la derecha)	ROR	Produce el desplazamiento de los bits del operando uno o más puestos a la derecha con la sustitución del MSB por el LSB	
Rotate left (rotación a la izquierda)	ROL	Produce el desplazamiento de los bits del operando uno o más puestos hacia la izquierda con la sustitución del LSB por el MSB	
Rotate right through carry (rotación a la izquierda con acarreo)	RORC	Produce el desplazamiento de los bits del operando uno o más puestos hacia la derecha. En el MSB se inserta el bit presente en el acarreo, mientras que el LSB se pone en el bit de acarreo	
Rotate left through carry (rotación a la derecha con acarreo)	ROLC	Produce el desplazamiento de los bits del operando uno o más puestos hacia la izquierda. En el LSB se inserta el contenido del bit de acarreo, mientras que el MSB se pone en el acarreo	
Test	TEST	Efectúa un examen (test) de los bits del operando, modificando exclusivamente los bits del Registro de Estado. Normalmente los bits de acarreo y desbordamiento se colocan a cero. El bit de medio acarreo no se modifica. 	

OPERACIONES DE AND Y TEST SOBRE EL RESULTADO



El código NOP significa No Operation. Cuando lo encuentra, el microprocesador no realiza ninguna acción (tiene un significado similar al REM del Basic).
En la figura se ha representado el equivalente Basic del programa, que sólo tiene valor indicativo, puesto que con el nombre simbólico A se entiende cualquier variable y no el acumulador, como sucede en el Assembler

rección o con el nombre simbólico. En el segundo caso es misión del Ensamblador traducir el nombre simbólico en una dirección, utilizando algunas normas que el programador debe insertar al principio de su programa. Por ejemplo, la instrucción

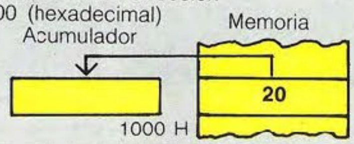
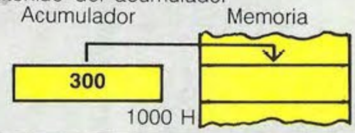
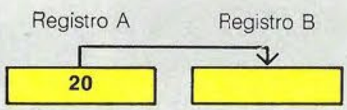
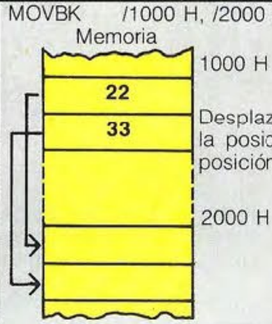
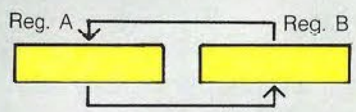
> ST A,/100H
> (memoriza el contenido de A en la celda que ha direccionado con el hexadecimal 100)

transfiere el contenido del acumulador en la celda de memoria cuya dirección es 100 hexadecimal* (direccionamiento absoluto).
En las páginas 926 y 927 se han relacionado las principales instrucciones de transferencia. Las instrucciones Move Block (MOVBK) y Move Multiple (MOVM) desplazan bloques de datos, así como bytes sencillos o dobles. Son muy útiles para el programador y pueden mejorar las prestaciones de una máquina, especialmente durante las operaciones de entrada/salida. La instrucción Exchange (XCH) activa un doble desplazamiento de datos, es decir un intercambio de los contenidos entre dos operandos. Por ejemplo, en algunas CPU es posible intercambiar los registros XL con la pareja de registros DE:

XCH DE,XL

* El número hexadecimal se especifica con la letra H.

INSTRUCCIONES DE TRANSFERENCIA DE DATOS

Instrucción	Código mnemónico	Descripción	Ejemplo
Load (carga de registro)	LD	El contenido de una posición de memoria especificada como origen se transfiere a un registro especificado como destino	LD A, /1000 H Carga en A el contenido de la celda de dirección 1000 (hexadecimal) 
Store (memorización)	ST	El contenido de un registro especificado como origen se transfiere a una posición de memoria especificada como destino	ST A, /1000 H Carga en la celda de dirección 1000 el contenido del acumulador 
Move (transferencia)	MOV	El contenido de un registro (posición de memoria) se transfiere a otro registro (posición de memoria)	MOV A, B A origen, B destino 
Move block (transferencia en bloque)	MOVBK	Esta instrucción produce la transferencia de un bloque de datos	MOVBK /1000 H, /2000 H, 2  Desplaza 2 bytes de la posición 1000 a la posición 2000
Move múltiple (transferencia múltiple)	MOVM	El contenido de una posición de memoria se copia en más posiciones	
Exchange (intercambio)	XCH	Los registros especificados se intercambian el contenido	XCH A, B Intercambio de contenido entre A y B 

INSTRUCCIONES DE TRANSFERENCIA DE DATOS

Instrucción	Código mnemónico	Descripción	Ejemplo
Input	IN	El dato presente en la puerta de entrada se transfiere a un registro o a una posición de memoria	IN A, (P1) El dato leido por la puerta de entrada P1 se pone en el acumulador Acumulador ← Puerta P1
Output	OUT	El contenido de un registro o de una posición de memoria se transfiere a una puerta de salida	OUT (P1), A El contenido del acumulador se envía a la puerta P1 Acumulador → Puerta 1
Clear	CLR	El operando especificado se pone a 0	CLR A El contenido del acumulador se pone a 0 Acumulador antes: 01011110 → Acumulador después: 00000000
Clear carry	CLRC	El bit de acarreo se pone a 0 (condición falsa o acarreo no existente)	CLRC El bit de acarreo se pone a 0 Estado antes: 1 (C Z O N H P) → Estado después: 0 (C Z O N H P)
Clear overflow	CLRV	El bit de desbordamiento se coloca a 1 (condición verdadera, desbordamiento existente)	CLRV Desbordamiento puesto a 0 Estado antes: Z = 1 (C Z O N H P) → Estado después: Z = 0 (C Z O N H P)
Set	SET	El operando especificado se llena con una serie de cifras 1	SET A El contenido del acumulador se pone a 1 Acumulador antes: 01011110 → Acumulador después: 11111111
Set carry	SETC	El bit de acarreo se pone a 1 (condición verdadera, acarreo existente)	SET C Acarreo puesto a 1 (C)←1 Estado antes: 0 (C Z O N H P) → Estado después: 1 (C Z O N H P)
Set overflow	SETV	El bit de desbordamiento se pone a 0 (condición falsa o desbordamiento no existente)	SET V Desbordamiento puesto a 1 (V)←1 Estado antes: Z = 0 (C Z O N H P) → Estado después: Z = 1 (C Z O N H P)

Generalmente, las instrucciones de intercambio utilizan como operando uno de los registros del microprocesador.
Las instrucciones de entrada/salida activan operaciones análogas a las transferencias de datos, con la diferencia de que la fuente (o el destino) es una puerta (de entrada o de salida) así como una posición de memoria. Por este motivo, algunos microprocesadores tratan las puertas de entrada/salida como posiciones de memoria y no prevén instrucciones específicas de I/O. El inconveniente que se encuentra con las operaciones de transferencia de memoria es la mayor lentitud con respecto a las instrucciones dedicadas. Un motivo por el que a veces no se proporcionan las instrucciones de I/O reside en el hecho de que para realizarlas son necesarias (además del código máquina que identifica la instrucción) adecuadas señales de control de I/O que utilizan una o más patillas del microprocesador. El número de patillas generalmente está limitado a 40 y pueden producirse dificultades de construcción.
Las instrucciones Clear se utilizan para poner a cero inmediata y rápidamente un registro, una celda de memoria o determinados bits de estado (Clear Carry o Clear Overflow).
Las instrucciones Set son dobles con respecto a las anteriores, y permiten cargar todos 1 en un registro o en una celda de memoria, además de poner a 1 un bit de Registro de Estado.
Estas instrucciones no son indispensables, puesto que es posible obtener el mismo resultado utilizando una o más instrucciones siempre presentes (Load o Store), pero tienen la ventaja de una ejecución muy rápida.

INSTRUCCIONES DE SALTO

Instrucción	Código mnemónico	Descripción
Branch if zero (salta si cero)	BZ	Salta si el resultado de la última operación realizada es 0, es decir, si el bit de cero del Registro de Estado es 1
Branch if not zero (salta si diferente de cero)	BNZ	Salta si el resultado de la última operación es diferente de 0, es decir si el bit de cero del Registro de Estado es 0
Branch if equal (salta si igual)	BE	Análoga a la Branch si no cero
Branch if not equal (salta si diferente)	BNE	Análoga a la Branch si no cero
Branch if carry (salta si hay acarreo)	BC	Salta si el resultado de la última operación tiene un acarreo (si el bit de acarreo del Reg. de Estado es 1)
Branch if not carry (salta si no hay acarreo)	BNC	Salta si el resultado de la última operación no tiene acarreo, es decir si el bit de acarreo del Registro de Estado es 0
Branch if positive (salta si positivo)	BP	Salta si el resultado de la última operación es positivo, es decir si el bit negativo (o sign) del Registro de Estado es 0
Branch if negativo (salta si negativo)	BN	Salta si el resultado de la última operación es negativo, (si el bit negativo [o sign] del Registro de Estado es 1)
Branch if overflow (salta si hay desbordamiento)	BV	Salta si el resultado de la última operación produce un desbordamiento, es decir si el bit de desbordamiento (V u O) del Registro de Estado es 1
Branch if not overflow (salta si no hay desbordamiento)	BNO	Salta si el resultado de la última operación no produce desbordamiento, es decir si el bit de desbordamiento (V u O) del Registro de Estado es 0
Branch if greater than (salta si mayor)	BGT	Salta si existe una relación «mayor», es decir si los bits del Registro de Estado verifican la siguiente relación lógica NORV OR (NOTZ) + (NOTN) OR (NOTV) OR (NOTZ) = 1

Instrucciones de bifurcación (Branch)

Una instrucción de bifurcación (o derivación) es una instrucción que fuerza al microprocesador a saltar de un punto a otro del programa. El salto interrumpe la ejecución secuencial normal y el calculador pasa a realizar una parte del programa que se inicia en una dirección especificada.
En otras palabras, la instrucción de bifurcación carga en el Contador de Programa la dirección de la posición a la que se quiere saltar, y el microprocesador continúa la ejecución a partir de la instrucción direccionada por el nuevo valor contenido en el Contador de Programa.
Un salto puede ser incondicionado o condicionado al producirse un cierto suceso. Una bifurcación (o jump) incondicionada que pone un nuevo valor en el PC es desaconsejable, puesto que altera el flujo lógico del programa sin motivo aparente, mientras que una bifurcación activada según que se verifique o no una cierta condición hace más lógico y legible el programa.
Este concepto volverá a verse cuando se hable de la programación estructurada, en la que se impone como regla general evitar las instrucciones de salto incondicionado.
Para condicionar un salto se usan los bits del Registro de Estado (flags de estado) que se modifican en función de los resultados de las diversas instrucciones.
Las instrucciones de bifurcación condicionada, indicadas en la tabla visible en estas dos páginas, permiten al ordenador repetir una misma secuencia de instrucción hasta que no se produce una cierta condición.
Utilizando este tipo de instrucción es posible tomar decisiones, ya que puede determinarse

Branch if greater than or equal (salta si es mayor o igual)	BGE	Salta si existe una relación de «mayor o igual», es decir si los bits del Registro de Estado verifican la relación lógica (NORV) AND [(NOTN)OR(NOTV)]
Branch if less than (salta si menor)	BLT	Salta si existe una relación de «menor», es decir si entre los bits del Registro de Estado es verdadera la relación lógica [NOR(NOTV)] AND [(NOTN) ORV] = 1
Branch if less than o equal (salta si menor o igual)	BLE	Salta si existe una relación de «menor o igual», es decir, si entre los bits del Registro de Estado se verifica la relación lógica Z AND [NOR(NOTV)] AND [(NOTN)OR V] = 1
Branch if higher (salta si más grande)	BH	Salta si existe una relación de «más grande», sin tener en cuenta los signos de los números, es decir (NOTC) OR (NOTZ) = 1
Branch if not higher (salta si no más grande)	BNH	Salta si existe una relación de «no más grande» entre dos cantidades sin signo, es decir si CANDZ = 1
Branch if lower (salta si es más pequeño)	BL	Salta si existe una relación de «más pequeño», sin tener en cuenta los signos de los números, es decir si C = 1. Análoga a la Branch if carry
Branch if not lower (salta si no más pequeño)	BNL	Salta si existe una relación de «no más pequeño» sin tener en cuenta los números, es decir, si C = 0. Análoga a la Branch if not carry
Branch if parity even (salta si paridad par)	BPE	Salta si el bit de paridad vale 1 (paridad par), es decir si el número de bits puestos a 1 es par
Branch if parity odd (salta si paridad impar)	BPO	Salta si el bit de paridad es 0 (paridad impar), es decir si el número de bits puestos a 1 es impar

qué instrucciones realizar en base a los datos recibidos o a los resultados obtenidos en una operación anterior.
Las condiciones más sencillas se basan en el valor directo de los bits de estado como el acarreo (BC, Branch if Carry) o el cero (BZ, Branch if Zero).
Generalmente hay disponibles tanto las instrucciones que comprueban si la condición es verdadera (branch if zero, salta si cero) como las dobles, que comprueban si la condición no es verdadera (branch if not zero, salta si no cero).
Por ejemplo, en el programa

```
LD A, 3      (carga el valor 3 en el acumulador)
CMP A, 3     (compara el acumulador con el valor 3)
BZ CERO      (salta a la etiqueta CERO si el bit de
             cero se pone a 1)
```

se realiza siempre el salto, puesto que la condición es siempre verdadera. En cambio, si la última instrucción fuese

```
BNZ NO CERO  (salta a la etiqueta NO CERO si
             el bit de cero se pone a 0)
```

la condición sería siempre falsa, y el salto nunca se produciría.
Otras instrucciones de salto están condicionadas como si fuesen precedidas de una instrucción de Compare (CMP), o sea de comparación; por tanto, es posible tener operaciones como Branch if Equal (BE, salta si igual), Branch if Greater Than (BGT, salta si mayor), Branch if Less Than (BLT, salta si menor).

Instrucciones de salto (Skip)

Las instrucciones de skip (literalmente salta por encima) son saltos condicionados en los que si la condición especificada se verifica, se salta la siguiente instrucción a la de skip.
No se especifica una etiqueta de llegada del salto: si la condición resulta verdadera, el programa salta solamente la siguiente instrucción y continúa secuencialmente.
Por ejemplo, en el siguiente programa

```
            LD A,/100 H   (carga en el acumulador el contenido
                          de la celda de dirección hexadecimal 100)
            SUB A,1       (resta 1 al acumulador)
            SKP           (salta la siguiente instrucción si el
                          resultado es positivo)
            B NEGATIVO    (salta a la etiqueta NEGATIVO)
POSITIVO    NOP           (rutina POSITIVO)
            . . .
            B FIN         (salta a la etiqueta FIN)
NEGATIVO    NOP           (rutina NEGATIVO)
            . . .
```

después de haber realizado el test (SKP) puede tenerse la ejecución de la siguiente instrucción (B NEGATIVO) o el salto a la POSITIVO NOP. Un equivalente en Basic puede ser el siguiente:

```
10 A = 16Λ2
20 A = A − 1
30 IF A > 0 GOTO 50
40 GOTO...
50 ............
```

El mismo programa puede escribirse también en la forma

```
            LD A,/100 H
            SUB A,1
            BN NEGATIVO   (salta a la etiqueta NEGATIVO si el
                          bit N vale uno)
POSITIVO    NOP
            . . .
NEGATIVO    NOP
            . . .
```

La instrucción de skip es útil cuando el código Assembler no prevé instrucciones de bifurcación. Efectivamente, es posible tener el mismo efecto de una instrucción de bifurcación utilizando una instrucción de skip y un salto incondicionado. La ventaja que se tiene en este último caso es la obtención de un código más compacto, formado por instrucciones más cortas. Generalmente, las instrucciones de skip condicionadas disponibles son análogas a las instrucciones de bifurcación, con la única diferencia que, si la condición se verifica, el control salta sobre la siguiente instrucción a la de skip, y prosigue en secuencia. Los códigos de esta clase de instrucciones son similares a los de la tabla de las



C O Rear/Grazia Neri-West Light

Aplicaciones al trabajo de gestión del ordenador personal Apple.

págs. 928 y 929, sustituyendo la letra B por SK (es decir SKip en lugar de Branch).

Instrucciones de llamada a subrutina y retorno

Las instrucciones de llamada a subrutina son instrucciones de salto particulares.
La sintaxis es del tipo:

CALL LABEL (salta incondicionalmente a la etiqueta LABEL)

La instrucción de llamada puede ser condicionada. Por ejemplo,

CALLZ LABEL (salta a LABEL si Z = 1)
CALLGT LABEL (salta a LABEL si mayor que...)

A diferencia de lo que sucede para las instrucciones de salto (branch o skip) el contenido del Contador de Programa se salva antes de ser sustituido por el nuevo valor. Sólo después de esta operación se procede a realizar las instrucciones que se encuentran en la dirección especificada mediante la etiqueta. Cuando se encuentra una instrucción Return (instrucción de retorno, código mnemónico RET), el contenido del Contador de Programa salvado anteriormente se recarga en el registro, y la instrucción del programa prosigue con la instrucción siguiente a la CALL.
También la instrucción Return puede ser condicionada, análogamente a lo que sucede para las instrucciones Call:

RETZ (retorna si el bit de cero vale 1)
RETGT (retorna si mayor que...)

Por tanto, la característica peculiar de estas instrucciones es la de salvar y recuperar el contenido del PC. Generalmente esta alteración se realiza mediante la pila, mientras que el modo en que opera el microprocesador depende del tipo de llamada. Si la llamada es condicionada y la posición especificada se produce, el contenido del Contador de Programa es «empujado» (push) a la pila; por tanto, el contenido de la celda de memoria que sigue al código CALL (o sea LABEL) se carga en el PC, y el control pasa a la instrucción que se encuentra en la nueva dirección, y viceversa, si la condición especificada

en la llamada no se cumple, se realiza la instrucción que sigue a la llamada condicionada. En caso de llamada incondicionada siempre se tiene el salto a la instrucción cuya dirección se especifica mediante la etiqueta.

En ambos casos, el valor de PC se recupera en el acto de la ejecución de la instrucción RET (Return); esta operación comporta la transferencia al PC de la dirección salvada anteriormente en la pila.

En las dos tablas de esta página se relacionan las instrucciones de llamada a subrutina y las de retorno.

También es posible llamar desde el interior de una subrutina otra subrutina. En estos casos, cada vez que se realiza CALL, el valor del Contador de Programa se salva en la pila, y análogamente, cada vez que se realiza una instrucción Return, el valor de la parte superior de la pila se recupera en el Contador de Programa.

Instrucciones varias

En la tabla de la página siguiente se han relacionado algunas instrucciones que suelen encontrarse en los lenguajes Assembler y que no pertenecen a ninguna de las clases examinadas anteriormente.

NOP. Cuando el microprocesador descodifica la instrucción NOP (No OPeration, ninguna operación) no se realiza ninguna función durante un ciclo máquina. Terminado este intervalo de tiempo se realiza la instrucción que sigue a la NOP. El efecto de la NOP sólo es el de retardar la ejecución de la siguiente instrucción y se utiliza precisamente para insertar comentarios.

INSTRUCCIONES DE LLAMADA A SUBRUTINAS

Instrucción	Código mnemónico	Descripción
Call (llama)	CALL	Salta a la etiqueta especificada salvando el contenido del PC en la pila
Call if zero (llama si cero)	CALLZ	Si el bit de cero es igual a 1, entonces salta a la etiqueta especificada salvando el contenido del PC en la pila
Call if carry (llama si hay acarreo)	CALLC	Si el bit de acarreo es 1, entonces salta a la etiqueta salvando el contenido del PC en la pila
Call if negative (llama si negativo)	CALLN	Si el bit de signo es igual a 1 salta a la etiqueta, salvando el contenido del PC en la pila

INSTRUCCIONES DE RETORNO

Instrucción	Código mnemónico	Descripción
Return (vuelve)	RET	Recupera el valor del PC si se había salvado en la pila
Return if zero (vuelve si cero)	RETZ	Recupera el valor del PC salvado en la pila sólo si el bit de cero es 1
Return if carry (vuelve si hay acarreo)	RETC	Recupera el valor del PC salvado en la pila sólo si el bit de acarreo es 1
Return if negative (vuelve si negativo)	RETN	Recupera el valor del PC salvado en la pila sólo si el bit de signo es 1 (valor negativo)

INSTRUCCIONES VARIAS

Instrucción	Código mnemónico	Descripción
No Operation (ninguna operación)	NOP	No se realiza ninguna operación, y el control pasa a la siguiente instrucción
Push (empuja)	PUSH	Los operandos especificados son «empujados» a la pila
Pop	POP	El primer elemento de la pila se pone en el operando especificado
Halt (para)	HALT	Bloquea la ejecución de las instrucciones hasta que no interviene un acontecimiento exterior (interrupt)
Wait (espera)	WAIT	Análoga a la anterior, pero la ejecución puede reemprenderse también con un acontecimiento interno
Break (interrumpe)	BRK	Permite la ejecución de un programa de interrupción
Adjust (adapta)	ADJ	El contenido del operando, generalmente del acumulador, se «reajusta» para representar el correcto resultado BCD. Para las operaciones en BCD es necesario sumar 6 para evitar las configuraciones no admitidas en el código
Enable Interrupt (habilita el interrupt)	EI	Habilita las instrucciones, es decir es posible interrumpir el programa
Disable Interrupt (Deshabilita el interrupt)	DI	Deshabilita las instrucciones, es decir no es posible interrumpir la ejecución del programa hasta que no se rehabilitan las instrucciones con la instrucción EI
Translate (traduce)	TR	Dada una tabla (por ejemplo la tabla BCD) se obtiene a través de un índice el valor correspondiente a un elemento de la tabla

PUSH y POP. Estas instrucciones permiten gestionar un área de memoria de pila.
A través de una PUSH es posible insertar un valor en la pila, mientras que con una POP es posible hacer «saltar fuera de la pila» el último valor depositado.

HALT y WAIT. Mediante las instrucciones HALT (detente) o WAIT (espera) es posible detener la ejecución del programa hasta que un acontecimiento externo, generalmente una interrupción, no desbloquea el microprocesador. Este tipo de instrucción permite, por ejemplo, suspender la ejecución hasta que no se esté seguro de que un periférico está correctamente sincronizado (por ejemplo para iniciar una transferencia de un bloque de datos).

Break. La instrucción BRK (interrumpe) permite lanzar uno de los programas interrupciones sin que se produzca la interrupción hardware. También en este caso, el contenido del Contador de Programa se salva.

Enable Interrupt y Disable Interrupt. También las instrucciones EI (Enable Interrupt, habilita la interrupción) y DI (Disable Interrupt, deshabilita la interrupción) se utilizan para gestionar las interrupciones. Mediante la instrucción DI es posible deshabilitar las interrupciones, que se reactivan con la instrucción EI. De este modo, ninguna causa externa puede interrumpir la ejecución del segmento de programa comprendido entre las dos instrucciones.
Se utilizan para efectuar con seguridad las ope-

raciones más delicadas, como la transferencia de bloques de datos.

Adjust y Translate. Las instrucciones Adjust (ADJ, adapta) y Translate (TR, traduce) efectúan operaciones numéricas particulares, como por ejemplo traducciones de codificación.

Las reglas de ensamblaje

Se definen como **reglas de ensamblaje** las instrucciones que normalmente se insertan en un programa Assembler como informaciones para el Ensamblador, con el objeto de controlar la correcta traducción del programa de los códigos mnemónicos al lenguaje máquina.
Estas reglas también se llaman **pseudooperaciones**, puesto que no generan un código máquina realizable por el microprocesador. Son precisamente comandos dados al programa Ensamblador para definir símbolos, preparar espacio para las variables, asignar las áreas de memoria para programas y datos, definir el final del programa y el formato del listado.
También las reglas de ensamblaje se representan con códigos mnemónicos del todo análogos a las de las instrucciones; sin embargo, éstos se analizan aparte, puesto que no encuentran correspondencia con códigos máquina en binario. Cada Ensamblador debe prever (incluir) subrutinas (en Assembler) que efectúan las acciones impuestas por las reglas. Algunas de las reglas más comúnmente utilizadas por los programas Assembler se compendian en la tabla de esta página.

REGLAS DE ENSAMBLAJE

Instrucción	Código mnemónico	Descripción	Ejemplo
Originado	ORG	El operando asociado se pone en el Contador de Programa y especifica la posición de memoria donde empezarán el programa, la subrutina o los datos	ORG # 100 El programa empezará en la posición 100
Equate	EQU	Asocia a un símbolo una constante, una dirección o una expresión	ALFA EQU 2 Donde haya ALFA se sustituirá por el valor 2
End	END	Informa al Ensamblador que ha llegado al final del programa	END
Page	PAGE	La lista del programa prosigue en una nueva página	PAGE
Title	TITLE	Hace proseguir la lista en una nueva página y se imprime la cabecera asociada al código TITLE	TITLE «Ejemplo». Va a la nueva página y escribe Ejemplo
Rescrive memcry	RES	Reserva un bloque de memoria para poder memorizar después datos	VET RES 50 Se han reservado 50 bytes a partir de la dirección llamada VET
Data	DATA	El Ensamblador llena las posiciones de memoria especificadas con el valor dado	MATR DATA 3 Todos los bytes de MATR se ponen = 3

El diagnóstico computerizado (1)

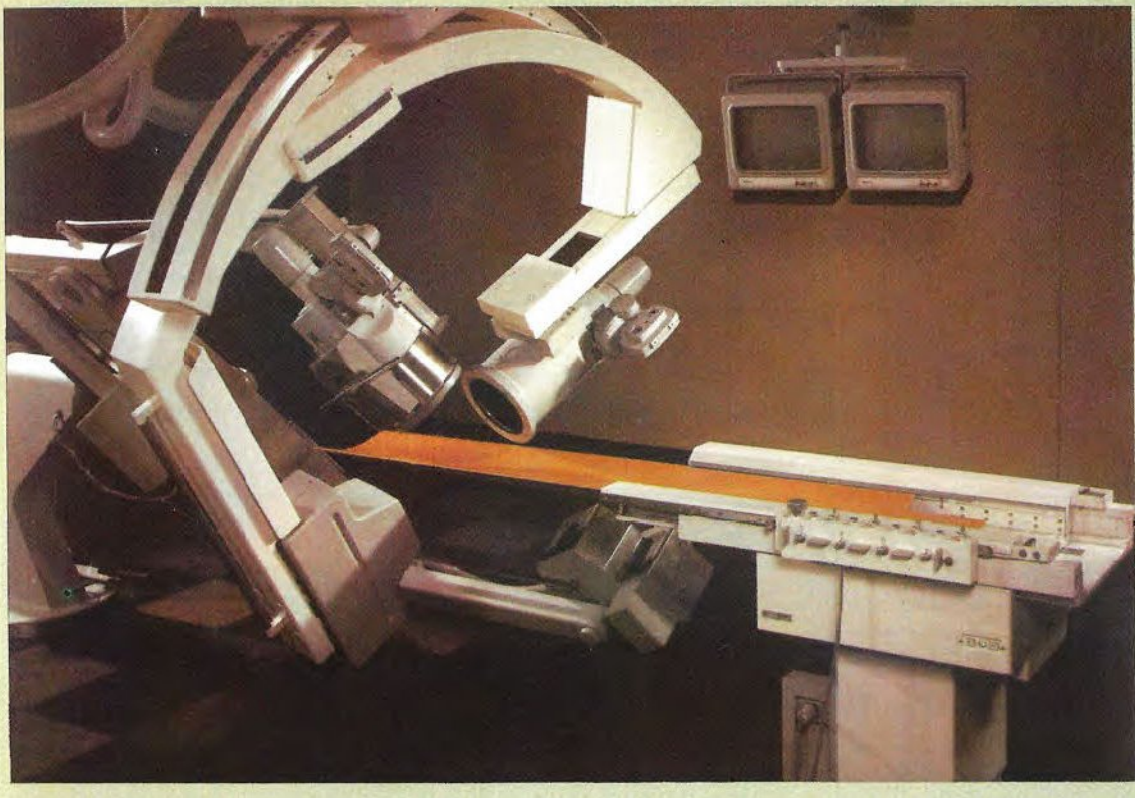
La extraordinaria evolución que la microelectrónica ha conocido en los últimos 20 años no ha dejado de producir sus efectos en todos los campos de la tecnología, con la masiva entrada del microprocesador en el control y en la realización de aparatos cada vez más sofisticados.
Los departamentos especialistas hospitalarios son quizá los lugares en que el advenimiento de la tecnología microelectrónica avanzada ha producido sus efectos más evidentes, y donde las nuevas conquistas tecnológicas se han aplicado de forma más difusa. La gama de los aparatos biomédicos que se benefician del control computerizado se orientan tanto al diagnóstico (con rayos X, nuclear, de ultrasonidos, termográfico) como a la terapia (electromedicina para la cardiología y para la monitorización de los pacientes en terapia intensiva). Las industrias que se ocupan de las aplicaciones biomédicas de la electrónica y de la informática actualmente están dotadas de importantes laboratorios de investigación aplicada, que permiten un continuo perfeccionamiento de los aparatos.
La evolución de la microelectrónica ha tenido como consecuencia otra rápida evolución de las aplicaciones biomédicas en general, que en pocos años ha dejado definitivamente obsoletas las grandes conquistas conseguidas desde el final del siglo XIX hasta los años sesenta. El mosaico de las aplicaciones tecnológicas en el sector biomédico se ha ampliado hasta el punto de crear el espacio para el nacimiento de una nueva ciencia aplicada, la bioingeniería, mientras los mismos aparatos de uso común, como por ejemplo los de diagnóstico por rayos X, han sufrido transformaciones para hacerlos más adaptables al desarrollo de análisis y a investigaciones extremadamente específicas.
En los primeros aparatos, la radiación se producía mediante una gran válvula (tubo de radiación) y se enviaba a una placa fotográfica o a una pantalla fluorescente, y entre el tubo y la placa se colocaba el sujeto a radiografiar. La intensa dosis de radiación a la que quedaban expuestos los pacientes no producía entonces una particular preocupación puesto que todavía no se sabía nada sobre sus efectos genéticos.
La realización de las primeras máquinas que podían efectuar radiografías de tórax constituyó una conquista sin precedentes, que permitió proporcionar un diagnóstico adecuado al problema de las afecciones pulmonares.
En los últimos años, las industrias de este sector han desarrollado una amplia gama de aparatos primarios y accesorios, que van desde la simple mesa horizontal a los dispositivos tomográficos pluridireccionales, a los cambios de película automáticos y a las reveladoras automáticas en línea. El grado de perfeccionamiento alcanzado en estos aparatos ha permitido reducir notablemente la dosis de radiación que se aplica a los pacientes y aumentar la definición y el significado de la imagen radiográfica.
La técnica de exploración por rayos X no se limita a la posibilidad de efectuar proyecciones normales del órgano en examen. La tomografía es una técnica que permite la visualización de una determinada capa anatómica imponiendo al tubo de radiación movimientos precisos (elípticos, circulares, lineales, epicicloidales) con el paciente dispuesto en la posición más adecuada. Los automatismos que controlan el movimiento de la superficie de la mesa permiten girar el sujeto incluso 180°, mientras que accesorios como los colimadores automáticos y los seriógrafos, a veces forman parte del equipo estándar de las máquinas. La aplicación del intensificador de brillo y del circuito cerrado de televisión, junto con la construcción de mesas completamente automáticas telecomandadas, permite hoy al operador trabajar en locales completamente aislados de los aparatos de radiología, resultado de significado fundamental para las exigencias de la radioprotección.
Un sistema de radiodiagnóstico de la última generación es el Diagnost 90, *un mastodonte de 1.650 kg de peso proyectado para satisfacer las exigencias de los laboratorios más grandes, en los que la cantidad de trabajo de rutina necesita características muy definidas de calidad de imagen, eficiencia y fiabilidad.*
El bastidor está suspendido flotante sobre el basamento, proyectado para ser colocado al abrigo de una pared. El basamento está encerrado en un armario metálico que contiene también el equipo electrónico. Las características de movimiento de exploración se controlan con un microprocesador, que permite inclinar el haz controlando automáticamente el paralaje y variar la distancia entre la fuente y la imagen. Todos los comandos para el accionamiento y para el control del posicionado del bastidor son duplica-

dos, y se encuentran tanto en el panel frontal del mismo bastidor como en una consola situada en posición remota.
El sistema está equipado con un cargador serie de placas provisto de control automático de la exposición y de la dosis para la fluoroscopia. La distancia que separa la película de la superficie de la mesa es de sólo 7 cm; esta característica permite trabajar en condiciones óptimas de geometría, eliminando totalmente los problemas asociados a las emisiones secundarias. El cargador puede aceptar placas de diversas medidas, cuyo centrado se produce automáticamente, mientras que el transporte en la posición de trabajo está controlado por un pulsador adecuado. Los programas de exposición prevén además la opción tomográfica.
El corazón de un sistema de radiodiagnóstico es el generador de rayos X. El sistema puede estar equipado con diversos generadores de potencia variable entre 50 y 100 kW, que permiten tiempos de exposición menores a 1 ms. El tubo de radiación está equipado con un colimador automático predispuesto para el empleo con intensificador de imágenes, y puede suministrar un haz exactamente cónico, eliminando así las enojosas radiaciones extrafocales. El colimador se abre o se cierra automáticamente, según el tipo de película insertado en el cargador serie, el programa de exposición seleccionado y la distancia fuente-película elegida.

Sistema de radiodiagnóstico computerizado Diagnost 90 de Philips.

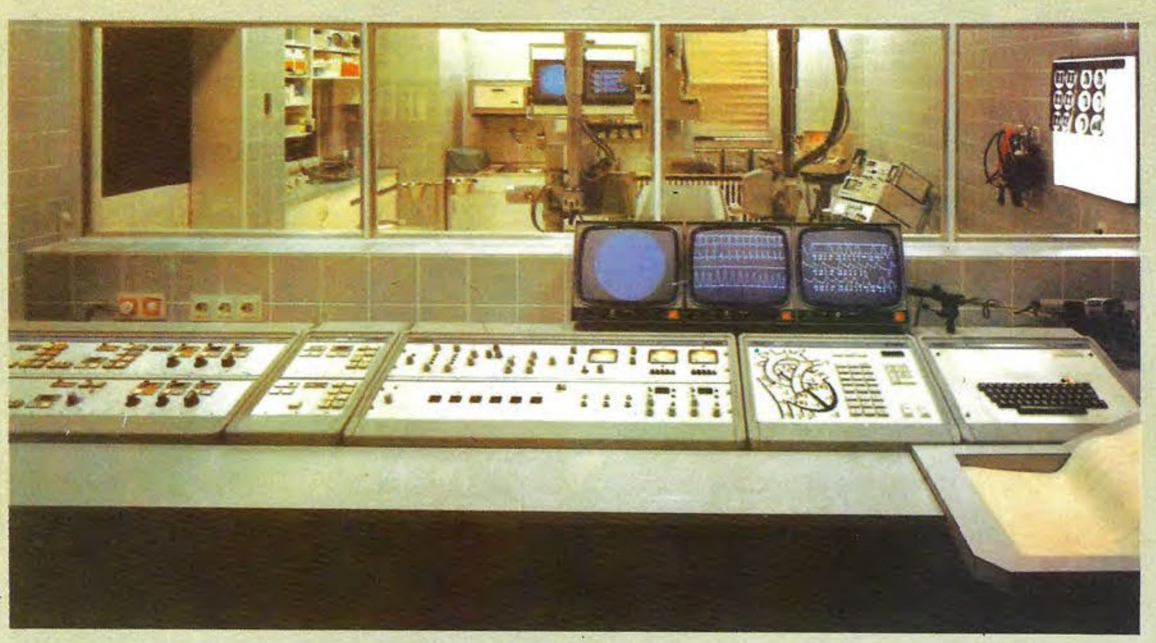


El cargador en serie está completado con un intensificador de imagen dotado de una pantalla de entrada de 9 pulgadas. Las elevadas características de absorción de la pantalla y las elevadas propiedades de transformación de la imagen permiten reducir notablemente las dosis necesarias para obtener imágenes fluoroscópicas suficientemente claras. La imagen fluoroscópica se visualiza a través de un circuito de televisión.
El control voltimétrico y el amperimétrico permiten mantener al nivel mínimo indispensable el flujo de radiación a través de la pantalla de entrada y, en consecuencia, minimizar también la dosis a la que se expone el paciente. La imagen fluoroscópica visualizada en el monitor puede registrarse automáticamente sobre película o papel sensible gracias a una fotocámara fluorográfica de 105 mm, dispuesta debajo del panel anterior, que incluye el cargador en serie. La fotocámara impresiona automáticamente en el margen de la imagen un código de identificación del paciente.
Las posibilidades del movimiento del sistema examinado son notables, y finalizan con aplicaciones particulares, como los exámenes radiológicos del aparato gastro-intestinal y la tomografía planar-paralela (es decir realizada sobre un plano paralelo al del bastidor). Para la ejecución de una rápida serie de proyecciones del esófago, por ejemplo, un programa adecuado combina el movimiento de exploración del tubo de radiación con el movimiento longitudinal del plano del bastidor, para realizar un movimiento relativo muy rápido (y por tanto, una velocidad de exploración muy elevada) sin molestias para el paciente. De este modo es posible trabajar a velocidades iguales a las del movimiento de deglución y seguir así paso a paso el movimiento del medio de contraste.
La tomografía y la zonografía planar-paralela necesitan la posibilidad de girar el tubo de radiación alrededor de un eje ortogonal al eje del cuerpo y paralelo al plano del bastidor. Gracias a este movimiento es posible cancelar los detalles que no interesan, evidenciando solamente los que corresponden a una zona restringida, aproximadamente plana, alrededor del eje de rotación. En los sistemas como el Diagnost 90, la opción tomográfica es completamente automatizada y puede seleccionarse apretando un pulsador de la consola.



Sistema de radiodiagnóstico computerizado Poly-Diagnost C. A la izquierda, en primer plano, es visible el brazo lateral.

Esta operación impone automáticamente una distancia fuente-imagen de 110 cm. Inicialmente, el tubo de radiación está posicionado en el eje perpendicular, de modo que permita el control del correcto apuntado a través de la imagen fluoroscópica y, por tanto, efectúa el movimiento rotatorio de exploración, volviendo automáticamente a la posición inicial.
El sistema de control integrado va a parar a una consola en la que los comandos y los indicadores están colocados según sectores separados, cada uno de los cuales está dispuesto para el control de las diversas modalidades de operación posibles.
Los circuitos de control de las diversas funciones están montados sobre tarjetas separadas y son supervisados por un microprocesador que comprende una memoria programada de sólo escritura (ROM). Las técnicas radiológicas de detección angiográfica han tenido, como los otros métodos de exploración con rayos X, progresos conseguidos mediante la gestión automática de los aparatos.
Dos sistemas angiográficos de la última generación son el Poly-Diagnost UPI *para la angiografía general y el* Poly-Diagnost C, *expresamente diseñado para la angiografía cardiaca.*
En el primer sistema, de tipo multidisciplinario, la característica principal que se observa a primera vista es la estructura del brazo en U, que permite girar 330° el tubo de radiación alrededor del cuerpo del paciente en observación. Esta posibilidad de rotación resulta indispensable para evidenciar los detalles más sutiles que caracterizan la imagen angiográfica. El peso total del aparato (proyector y bancada) es de cerca de una tonelada.
En los dos extremos del brazo en U hay instalados el tubo de radiación y el grupo intensificador de imagen/cargador de película.
El control de todo el aparato puede programarse sobre fichas perforadas, así como puede estructurarse el movimiento de la fuente de la forma más adecuada para resolver cada exigencia particular de exploración. El sistema Poly Diagnost C *se dedica, como se ha dicho, principalmente a la cardio-angiografía. Su estructura puede verse en la fotografía de arriba donde se observan los detalles constructivos del brazo en paralelogramo articulado.*



Philips

Consola de control y de maniobra del sistema Poly-Diagnost C.

Esta solución permite la ejecución de proyecciones de acuerdo con cualquier ángulo de inclinación, lo cual asume una notable importancia en el examen de órganos intensamente vascularizados como el músculo cardiaco. El posicionamiento inicial del brazo en paralelogramo se realiza esta vez manualmente y puede obtenerse con una ligera acción sobre la manija. Otra característica peculiar es la cama enteramente libre, que permite la exploración de cualquier región anatómica sin que sea necesario desplazar el paciente. La configuración del sistema puede completarse con un segundo brazo lateral, que en unión del primero, permite la exploración biplanar simultánea del corazón. Ambos brazos giran de forma síncrona bajo el control del microprocesador central.

El sistema de funcionamiento de los aparatos radiográficos es del todo similar al de los aparatos de radiodiagnóstico de uso general. El control automático de la distancia entre el tubo de radiación y el sujeto examinado elimina cualquier peligro de contacto, que sería particularmente peligroso especialmente en el caso de aplicación de cateterismos.

Como puede observarse en la fotografía de esta página, la consola de maniobra y de control de un aparato de este tipo no tiene nada que envidiar al panel de control de una estación ferroviaria. Todo el sistema está gestionado por un ordenador que procesa en tiempo real todos los datos correspondientes al particular examen realizado.

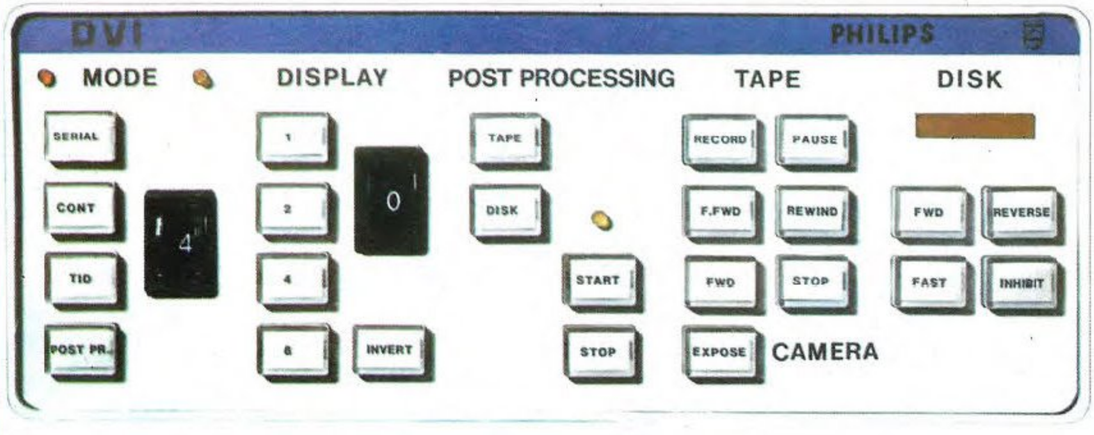
El último hallazgo de la tecnología para la resolución del sistema vascular mediante aparatos de rayos X es la llamada radiografía vascular digitalizada. *Se trata de una nueva metodología de proceso numérico de la imagen radiográfica que promete convertirse en la técnica estándar de exploración del próximo futuro, dada la relativa sencillez de los principios y de los aparatos en que se basa.*

El esquema de principio del sistema de proceso se ha representado en la pág. 940. Existen tres niveles diferentes de proceso numérico de la señal procedente del aparato radiológico, cada uno de los cuales está representado por el bloque «algoritmos» y el operador puede seleccionarlo apretando uno de los tres pulsadores superiores de la sección MODE del panel del control. A ello se añade un cuarto tipo de proceso (post-processing) *que puede intervenir en los datos registrados de cada uno de los tres procesos principales.*

La señal analógica procedente de la pantalla fluoroscópica y del intensificador de imagen del aparato de exploración se envía a través del interfaz a un amplificador logarítmico. El digitalizador (convertidor analógico-digital) convierte la salida del amplificador en señales numéricas que se memorizan. Por tanto, el contenido de la memoria se procesa en tiempo real con uno de

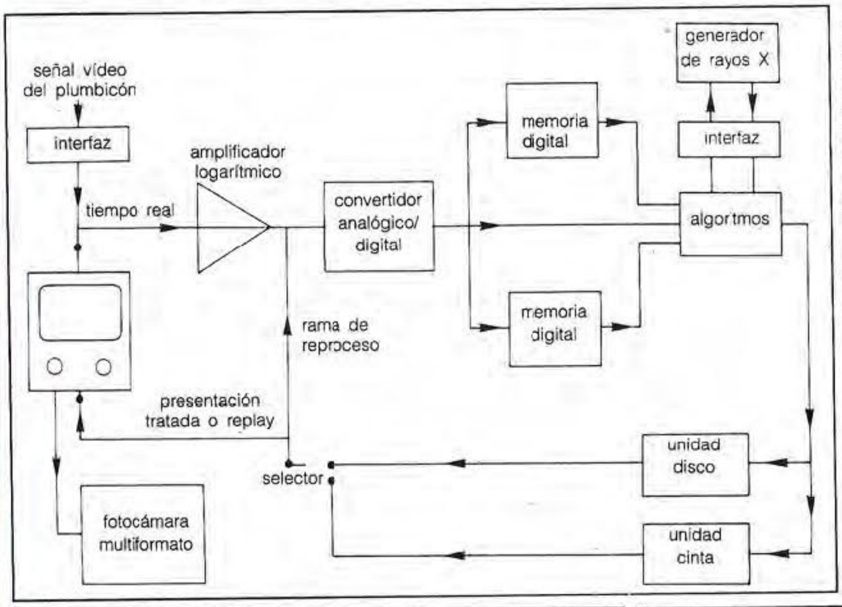
los tres algoritmos fundamentales, y la salida, que se registra en disco o cinta, se presenta simultáneamente en el monitor.
El primero de los tres algoritmos de proceso se denomina serial imaging mode y se ha representado de forma esquematizada en la figura A. Después de la introducción en la vena del medio normal de contraste, pero antes de que la arteria o el vaso examinado se hagan opacos, se toma una imagen fluoroscópica del campo, que después servirá como máscara sustractiva. Efectivamente, esta imagen se resta con métodos electrónicos de una serie de imágenes tomadas sucesivamente a intervalos de cerca de 1 s (la duración de este intervalo de tiempo puede variarse). El resultado de cada operación de resta es una nueva imagen en la que el fondo (los órganos que no interesan) queda cada vez más debilitado, hasta visualizar solamente el vaso objeto de investigación. El serial imaging mode se ha aplicado con éxito al examen de la carótida, de los vasos periféricos y de las arterias renales y abdominales.
La segunda posibilidad de proceso es el continuous imaging mode. En este caso (fig. B), forma la manera dicha anteriormente una máscara fluoroscópica antes de la inyección del medio de contraste, que seguidamente se resta de forma continua a la imagen fluoroscópica del detalle hecho opaco, a intervalos de 16,6 ms. La imagen producida se presenta en tiempo real en el monitor y simultáneamente se memoriza en disco o cinta. El proceso podrá reclamarse después sobre el monitor a partir del soporte magnético. Esta técnica se presta de forma particular para evidenciar los movimientos de la pared ventricular, las valvulopatías y las arterias pulmonares.
El último algoritmo de proceso se denomina time interval difference mode (TID) y consiste en la resta de imágenes producidas a intervalos constantes por el continuous imaging mode y registradas en memorias. En el esquema de la figura C se presenta la producción de tres imágenes por resta de cada tercera imagen de la serie registrada previamente.
El postprocessing mode (fig. D) es fundamentalmente un sistema de proceso «off-line» que, por tanto, no necesita la permanencia del paciente bajo el aparato radiográfico, y permite procesar las imágenes anteriormente registradas aplicando una u otra de las posibilidades indicadas. Las posibilidades ofrecidas por la radiografía vascular digitalizada son muchas. En primer lugar se subraya la relativa no impasividad del método, asociada a una resolución de la imagen procesada mejor que la que es posible obtener con los tradicionales sistemas angiográficos. Además, las imágenes procesadas con el sistema sustractivo son producidas en tiempo real, y es posible tener en cualquier momento una copia sobre papel de lo que se ha presentado en el monitor gracias a una fotocámara multiformato conectada al aparato.
Finalmente debe observarse que el sistema digital puede aceptar a la entrada la imagen fluoroscópica que se produce con un aparato radiológico cualquiera. Estas circunstancias dejan prever para el próximo futuro una amplia difusión de este nuevo método.

Panel de los comandos y de los controles del sistema Digital Vascular Imaging.

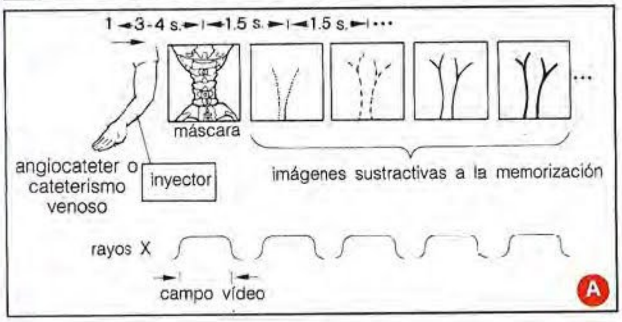


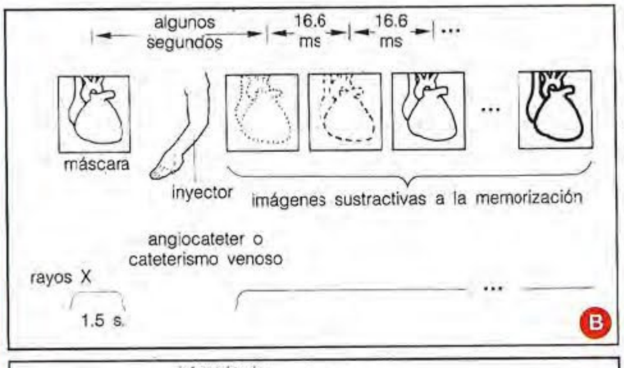
Philips

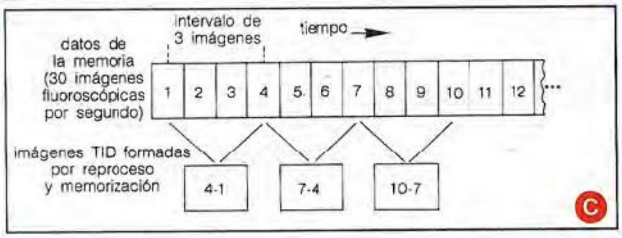
ESQUEMAS Y MODOS DE FUNCIONAMIENTO DEL SISTEMA DIGITAL VASCULAR IMAGING

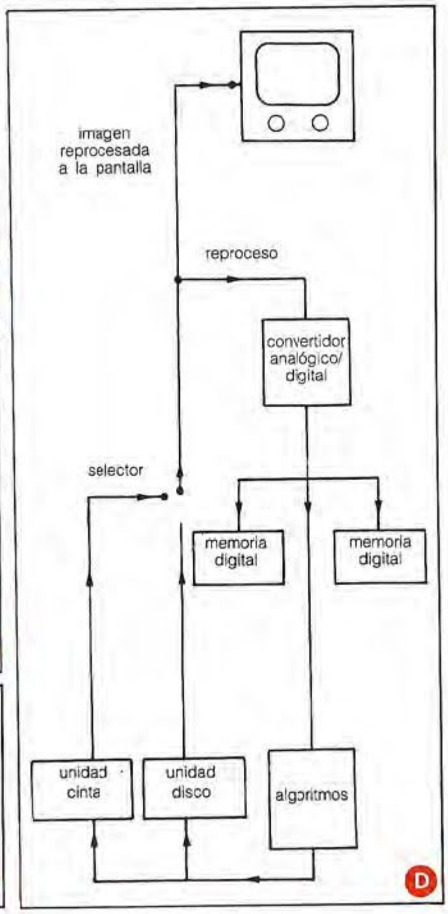


A la izquierda puede verse el esquema de bloques funcionales del aparato. En A, el serial imaging mode, en B el continuous imaging mode, en C el time interval difference mode (TID) y en D el post processing mode, que puede intervenir sobre las imágenes procesadas con uno de los algoritmos anteriores.









La regla ORG (código mnemónico para ORIGINATE) permite definir una dirección inicial de memorización para un programa, un subprograma o un conjunto de datos. La ORG debe ir seguida de un operando, normalmente de tipo inmediato. Por ejemplo, escribiendo

```
ORG 100
LD A,ALFA
. . .
```

se consigue que la parte del programa a partir de LD A,ALFA se cargue en memoria a partir de la dirección 100 en adelante.
Con el uso de la regla ORG se evita que el programa principal, las subrutinas y las áreas de datos se superpongan en la memoria o vayan a superponerse en áreas de memorias utilizadas normalmente por el Sistema Operativo. En un programa Assembler pueden haber presentes más reglas ORG.
La regla EQU (código mnemónico para EQUATE) permite definir el nombre de una variable y asignarle un valor numérico. Cada vez que en el programa se reclame la variable, ésta sustituirá la constante numérica asignada.
En esta regla, el nombre de la variable normalmente está colocado en el campo etiqueta de la instrucción, como muestra el siguiente ejemplo:

```
COUNT     EQU 1
ALTA      EQU 20
          LD A,ALFA
          . . .
```

Con la asignación ALFA EQU 20, la instrucción LD A,ALFA se hace equivalente en código máquina a la LD A,20.
Normalmente, las reglas EQU están posicionadas al principio del programa, como documentación y para favorecer la legibilidad.
La regla RES (código mnemónico para RESERVE) reserva espacio RAM para las variables, y asigna un nombre a la primera dirección del área reservada. Por ejemplo, la regla

```
BUFF      RES 50
```

crea un área de memoria de 50 posiciones y asigna a la primera de ellas el nombre BUFF. De este modo, el acceso al área de memoria reservada puede realizarse haciendo referencia al nombre BUFF. Por ejemplo, con la instrucción



J. Pickerell/Marka

Una sala de máquinas de General Electric Co.

```
LD A, BUFF + 10
```

se carga en el registro A el contenido de la décima posición de memoria a partir de la posición de nombre simbólico BUFF, creada con la regla RES.
Finalmente, la regla DATA permite que el programa tenga acceso a tablas o constantes.
Normalmente, los valores designados mediante una instrucción DATA permanecen en memoria, mientras que la instrucción RES (RESERVE) sólo reserva el espacio para las variables.
Mediante la regla RES es posible reservar posiciones de memoria, para después llenarlas con los valores oportunos.
Por tanto, puede escribirse

```
BETA      DATA 2
```

y esto significa que la primera celda de memoria libre se llamará de ahora en adelante BETA (dirección) y contendrá el valor 2. Análogamente, la regla

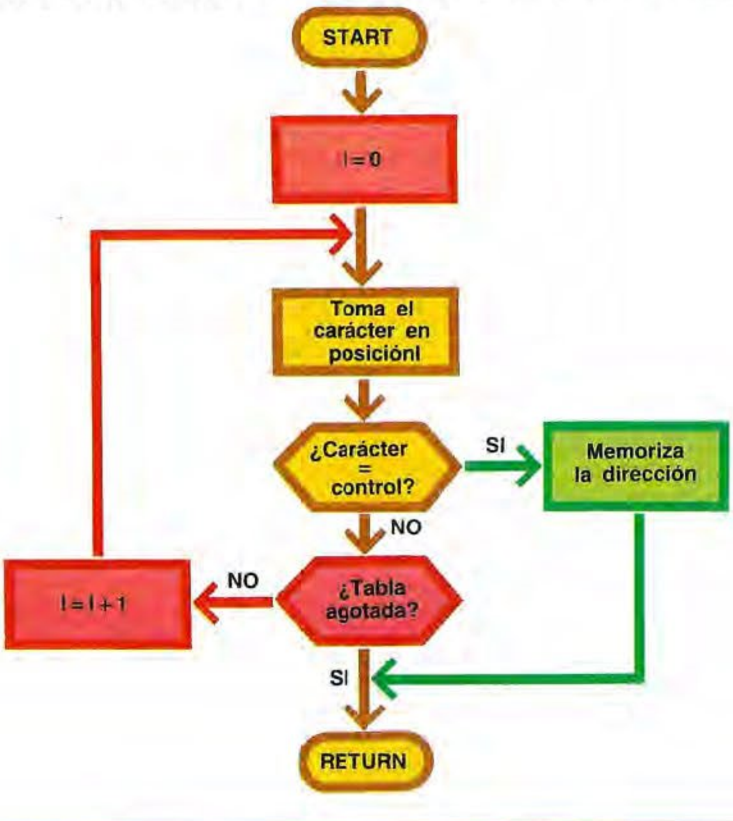
```
GAMMA     DATA 3,4,5,6,7
```

creará una tabla de cinco elementos, en la que el primer elemento tendrá la dirección GAMMA y contendrá el valor 3; en los siguientes estarán memorizados los otros valores (4, 5, 6, 7).

Ejemplo de programación en Assembler

Como puede deducirse de lo dicho en los párrafos anteriores, el lenguaje Assembler es sin duda el más complejo de los lenguajes simbólicos de alto nivel como el Basic; sin embargo, a pesar de la existencia de tantos lenguajes, todavía hoy resulta insustituible para resolver determinados problemas de programación.
Su estrecha dependencia al funcionamiento real del microprocesador necesita, por parte del programador, una fase de análisis del problema mucho más sofisticada. Utilizando el lenguaje Assembler ya no es suficiente el desarrollo del diagrama de flujo de un procedimiento para la obtención de una sucesión de operaciones complejas (entrada de datos, procesos con varias funciones, presentación de los resultados), sino que hay que descender al detalle de cómo se realiza cada operación sencilla en el interior de la máquina.
En otras palabras, y para poner un ejemplo concreto, un diagrama de flujo en base al cual es posible detallar un programa Basic no es suficiente para permitir, en general, el detalle del mismo programa en Assembler, a menos que el programador tenga una gran experiencia. Es necesario que los bloques funcionales sencillos sean detallados posteriormente, hasta alcanzar una sucesión de operaciones que sean verdaderamente elementales (es decir explícitamente previstas) en el lenguaje Assembler.
Es evidente que, como sucede para cada lenguaje de programación, la experiencia permitirá hacer hasta un cierto punto diagramas de flujo del máximo detalle; así, el programador podrá explicitar directamente en la fase de detalle un determinado bloque funcional.
A título de ejemplo examinaremos ahora un sencillo ejemplo de programación en Assembler. El objeto será el de obtener una subrutina que realice la búsqueda de un carácter en una tabla. El diagrama de flujo de la lógica de búsqueda se indica abajo y, como puede verse, es muy sencillo.

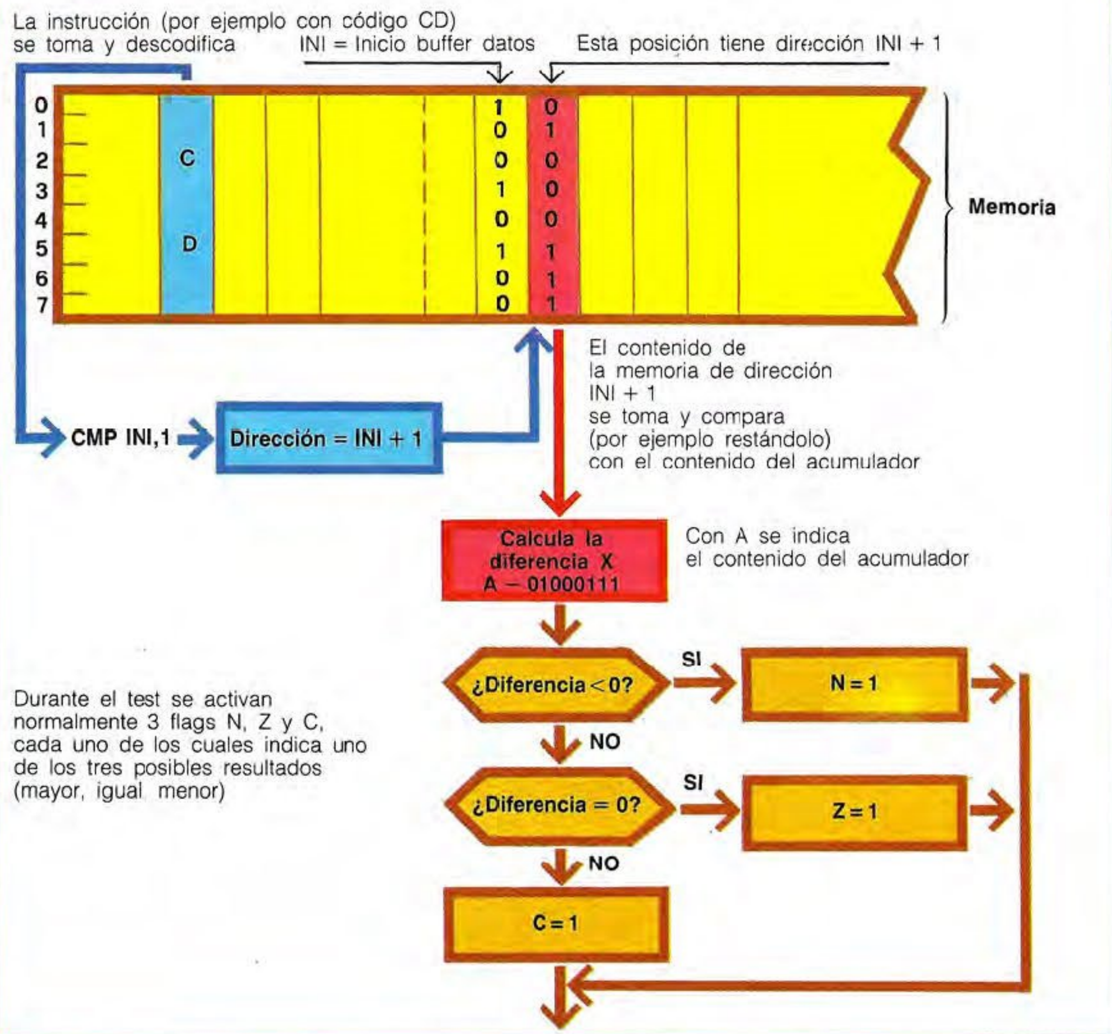
DIAGRAMA DE FLUJO DE BUSQUEDA EN UNA TABLA



Supongamos ahora que hemos cargado un cierto número de caracteres en una serie de posiciones de memorias contiguas (buffer), a partir de la dirección INI, y que se ha memorizado el carácter de control en la dirección CH. Sea además NC el número de los caracteres de la tabla. Nuestro programa deberá utilizar dos registros. En el registro X memorizaremos el puntero en el interior del buffer correspondiente a la dirección INI. El primer carácter está en la posición INI, el segundo en la posición INI + 1 etc.; en general un carácter genérico se encuentra en la posición INI + X, con X que va de 0 a NC-1. Durante el desarrollo del programa, el acumulador A contendrá el carácter de comparación y, al final, el puntero al elemento del buffer que satisface la igualdad. Examinaremos ahora la estructura del programa, recordando que los códigos mnemónicos de las instrucciones utilizadas dependen del microprocesador empleado.

	LDX # 0	Inicializa el registro X al valor 0
	LDA CH	Toma de la posición CH el carácter de comparación y lo pone en el acumulador
	CMP INI, X	(CMP dirección, incremento). Es una forma particular específica sólo de algunos microprocesadores. El significado es: toma el contenido de la memoria apuntada por dirección + incremento y compara el dato que encuentres con el contenido del acumulador. Si son iguales pon al valor 1 el flag Z (ver gráfico de la pág. 944). En la forma indicada, la instrucción compara el contenido de A con el de la memoria INI + X. En este punto del programa X = 0, por tanto se compara el contenido de la posición INI, o sea el primer elemento del buffer. Si la comparación es positiva, el flag Z se pone a 1.
	BEQ HALLADO	El código BEQ (branch if equal) activa un salto a la posición específica (de nombre HALLADO) si Z = 1, de otra manera (no hallado) se prosigue en secuencia
LOOP	**INX**	Incrementa X para tomar otro carácter del buffer
	CMP INI,X	Realiza la comparación ya descrita. antes
	BEQ HALLADO	Salta a HALLADO si Z = 1
	STX SALVA	Deposita el contenido de X en la memoria SALVA. De este modo, el registro X puede utilizarse provisionalmente para otras finalidades; el contenido original se recupera cargando SALVA
	CPX NC	Compara X con la longitud del buffer: si son iguales pone Z = 1
	BEQ NONT	Salta a NONT (no hallado)
	LDX SALVA	Recarga en X el valor que X tenía antes del test (es el puntero al último dato examinado)
	JMP LOOP	Salta a la posición LOOP, iniciando un nuevo ciclo de control
HALLADO	**NOP**	Punto de entrada en el caso de carácter hallado. El código NOP no tiene ningún efecto; es uno de los modos de escribir un comentario. El registro X contiene la posición (a partir de 0) en que se encuentra el elemento del buffer de datos

ESQUEMA LOGICO DE LA INSTRUCCION CMP



		igual al de control (no se consideran elementos sucesivos, incluso si satisfacen la igualdad)
	TXA	El contenido de X se transfiere al acumulador. En la salida, A contiene la posición en que se encuentra el elemento del buffer igual al carácter de control
NONT	**NOP**	No hallado (comentario)
	RTS	Retorno. Almacena el contenido del PC en la pila y lo incrementa en 1; en otras palabras, predispone para realizar la instrucción siguiente a la llamada de subrutina (JSR o RJP según el microprocesador utilizado).

El Assembler del Rockwell 6502

El microprocesador Rockwell 6502 ha tenido un gran éxito y en la actualidad existe, en varias versiones, en diferentes ordenadores personales. Es posible encontrarlo, por ejemplo, en las máquinas Commodore (VIC-20 y 64) y en el Apple II.

La disposición cualitativa de las patillas del microprocesador puede verse en la figura de abajo, mientras que en la figura de la pág. 946 puede verse la estructura interna y la configuración del registro de estado.

En el 6502, además del acumulador, hay los registros X, Y y S. Los registros X y Y se utilizan como índices, mientras que el registro S se utiliza para controlar la pila. Ésta contiene la dirección de la primera celda disponible de la pila, que en el 6502 va de la dirección física 256 a la dirección 511, comprendiendo así 255 bytes.

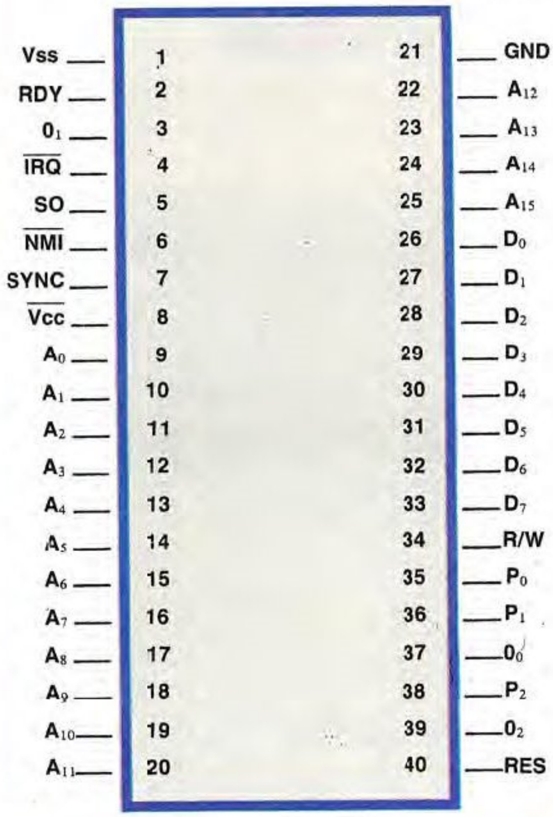
El registro de estado, registro P, comprende los siguientes flags:

- **N** Bit del signo (1 si negativo)
- **V** Bit de desbordamiento (1 si desbordamiento)
- **B** Bit de interrupción software (puesto a 1 si pide interrupción)
- **D** Bit de uso del código BCD (puesto a 1 cuando se usa la matemática BCD)
- **I** Bit de interrupción hardware (puesto a 1 cuando se deshabilitan las interrupciones)
- **Z** Bit de cero (puesto a 1 si la última operación ha dado resultado cero)
- **C** Bit de acarreo (puesto a 1 si hay una condición de acarreo; este flag se utiliza también en las instrucciones de desplazamiento o rotación)

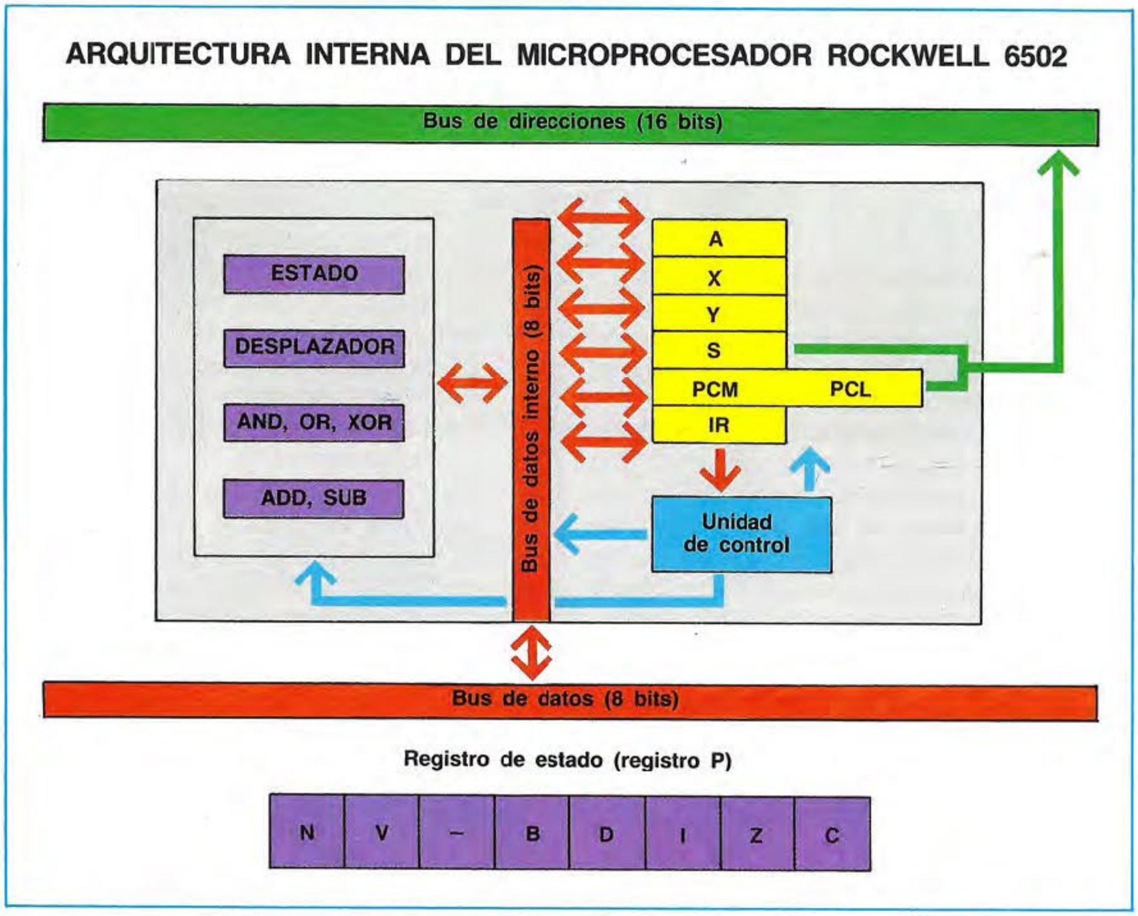
En la tabla de la pág. 947 se han indicado los códigos mnemónicos de las instrucciones del 6502 comparadas con los códigos estándar según la propuesta de la IEEE Task P694/011.

Puede observarse que las instrucciones son especializadas, o sea operan directamente sobre los registros presentes (A, X, Y, S).

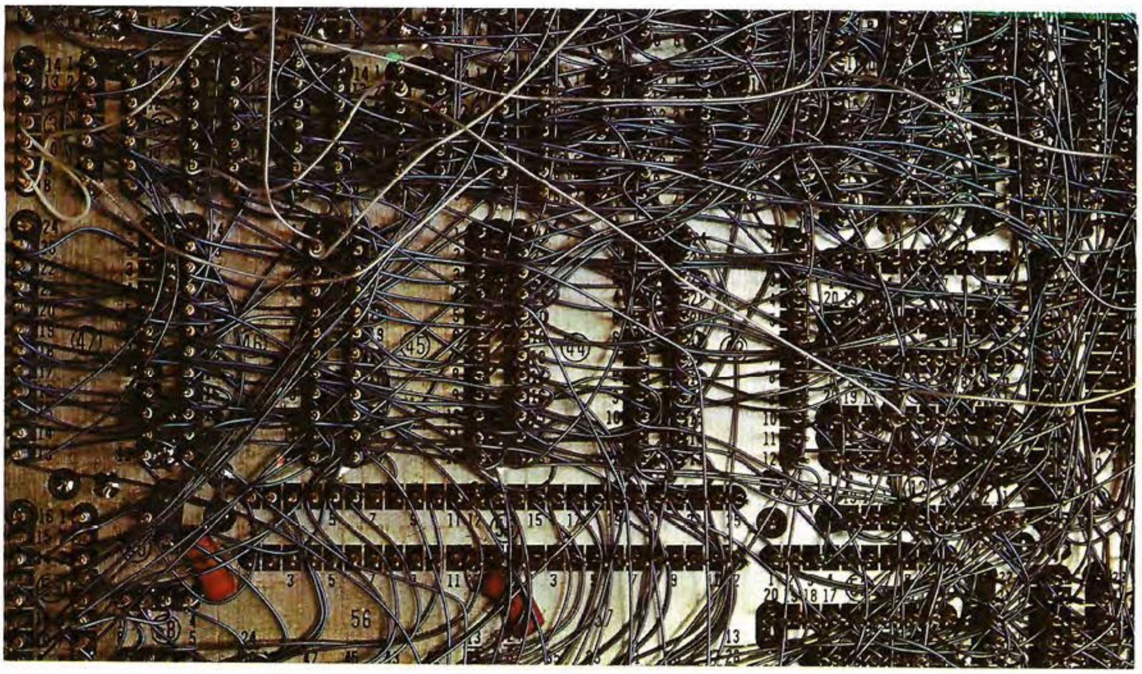
PATILLAS DEL MICROPROCESADOR ROCKWELL 6502



Líneas de dirección:	$A_0 \div A_{15}$
Líneas de datos:	$D_0 \div D_7$
Líneas de alimentación:	Vss, Vcc, GND (±5 volt)
Líneas de control:	
Señales de reloj	0_0, 0_1, 0_2
Señales de reset	RES
Controla el flag de desbordamiento	SO
Señales de sincronismo	SYNC, RDY
Señales de lectura/escritura	R/W
Líneas de interrupción	IRQ, NMI



El intrincado sistema de las conexiones externas de un ordenador.



J. Pickerell/Marka

INSTRUCCIONES ASSEMBLER DEL MICROPROCESADOR ROCKWELL 6502

Instrucción	Código mnemónico IEEE	Código mnemónico 6502
Instrucciones aritméticas		
Add with carry	ADDC	ADC
Subtract with carry	SUBC	SBC
Increment	INC	INC
Increment X		INX
Increment Y		INY
Decrement	DEC	DEC
Decrement X		DEX
Decrement Y		DEY
Compare to A	CMP	CMP
Compare to X		CPX
Compare to Y		CPY
Instrucciones lógicas		
AND	AND	AND
OR	OR	ORA
Exclusive OR	XOR	EOR
Shift Right	SHR	LSR
Arithmetic Shift Left		ASL
Rotate Right	ROR	ROR
Rotate Left	ROL	ROL
Test Bit	TEST	BIT
Instrucciones de transferencia de datos		
Load A	LD	LDA
Load X		LDX
Load Y		LDY
Store A	ST	STA
Store X		STX
Store Y		STY
Clear Carry	CLR	CLC
Clear Decimal		CLD
Clear Interrupt		CLI
Clear Overflow		CLV
Set Carry	SETC	SEC

Instrucción	Código mnemónico IEEE	Código mnemónico 6502
Set Decimal		SED
Set Interrupt Mode		SEI
Transfer A to X	MOV	TAX
Transfer A to Y		TAY
Transfer SP to X		TSX
Transfer X to A		TXA
Transfer X to SP		TXS
Transfer Y to A		TYA
Instrucciones de salto		
Branch if Zero	BZ	BEQ
Branch if Not Zero	BNZ	BNE
Branch if Negative	BN	BNI
Branch if Positive	BP	BPL
Branch if Carry	BC	BCS
Branch if No Carry	BNC	BCC
Branch if Overflow Clear	BNV	BVC
Branch if Overflow Set	BV	BVS
Jump		JMP
Instrucciones de llamada a subrutina		
Jump to Subroutine	CALL	JSR
Instrucciones de retorno		
Return from Subroutine	RET	RTS
Return from Interrupt		RTI
Instrucciones varias		
No Operation	NOP	NOP
Push A	PUSH	PHA
Push P (status)		PHP
Pop A	POP	PLA
Pop P (status)		PLP
Break	BRK	BRK

El Assembler del Zilog Z80

El Zilog Z80 es uno de los más sofisticados microprocesadores de 8 bits. Se desarrolló para ser compatible con el Intel 8080, pero con características superiores. Actualmente se emplea en los microordenadores más potentes y con él se ha desarrollado el sistema operativo CP/M (Control Programming for Microcomputer).

La disposición cualitativa de las patillas del microprocesador se indica en el gráfico de la página 948, donde también se indican las misiones de las líneas de conexión más importantes.

La arquitectura interna del Zilog Z80 está esquematizada en la pág. 949, donde se indica también la estructura del registro de estado.

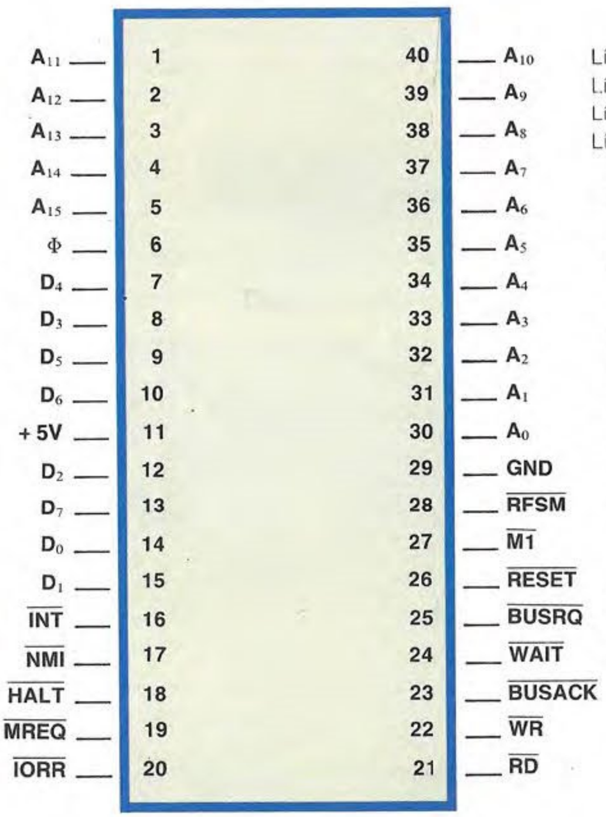
Internamente, el microprocesador Z80 es bastante complejo. Está compuesto por dos acumuladores (A, A') y por dos pares de 6 registros (BC, DE, HL, B'C', D'E', H'L'). Existen además dos registros índices (IX, IY), una pila puntero (SP), un contador de programa (PC), el registro de instrucciones (IR) y dos registros especializados I y R. El primero se utiliza con las instruc-

ciones, mientras que el segundo sirve para gestionar los «refrescos» de la memoria.
En la unidad lógico-aritmética hay dos registros temporales: TMP y Acc. Temp. utilizados por el acumulador para realizar los cálculos.
Los registros de estado son dos, idénticos, llamados F y F' y contienen los siguientes bits:

S Bit del signo
Z Bit de cero
H Bit de medio acarreo (BCD)
P/V Bit de paridad o desbordamiento (según el valor de los otros bits)
N Bit de resta, usado por el sistema en las operaciones BCD
C Bit de acarreo

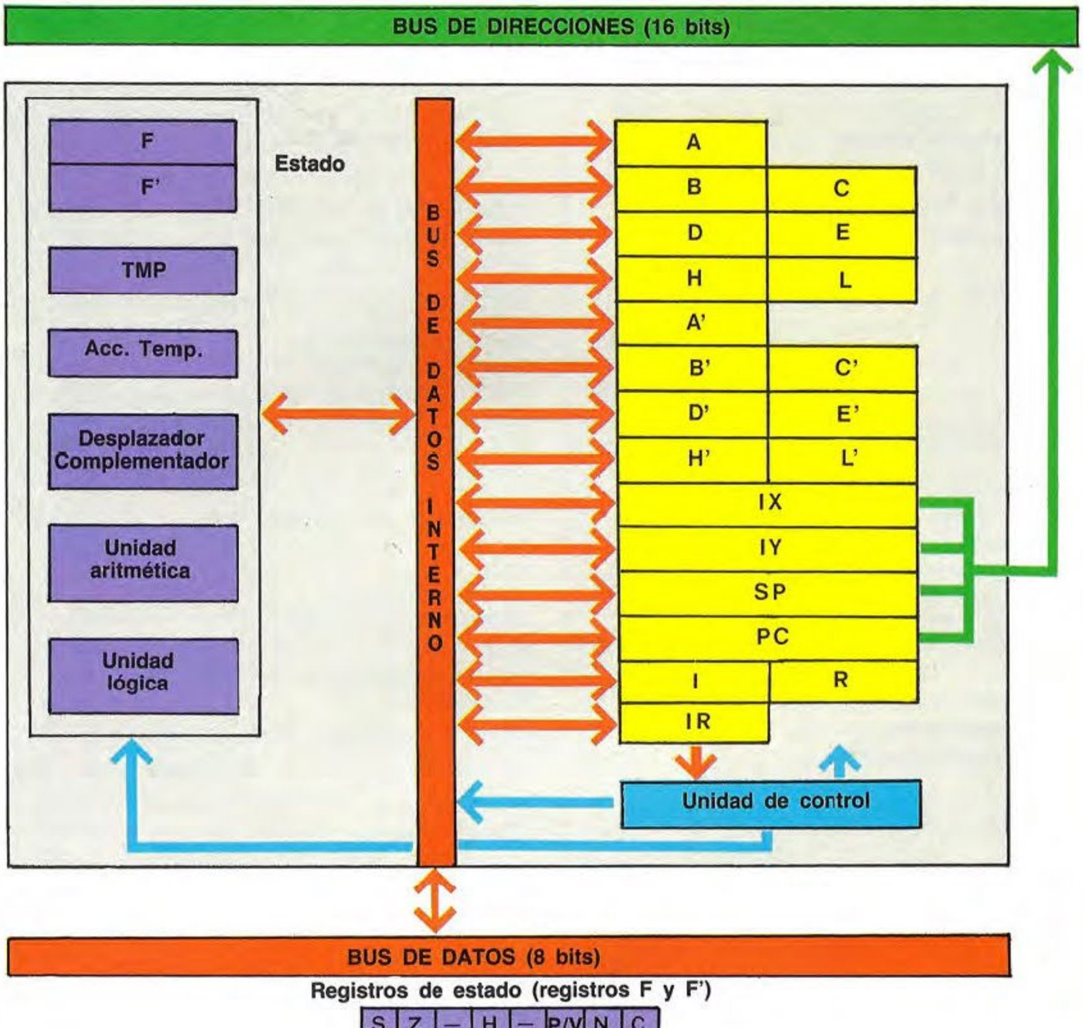
En la tabla de las págs. 949 y 950 se han indicado los códigos mnemónicos de las instrucciones Zilog Z80 comparados con los códigos estándar IEEE. Como puede verse, el conjunto de las instrucciones disponibles es muy amplio, y esto permite una notable flexibilidad en el empleo de este microprocesador.

PATILLAS DEL MICROPROCESADOR Z 80



Líneas de dirección:	$A_0 \div A_{15}$
Líneas de datos:	$D_0 \div D_7$
Líneas de alimentación:	+5 V, GND
Líneas de control:	
Reloj de la CPU:	Φ
Ciclo máquina de búsqueda:	$\overline{M1}$
Indica que la CPU está accediendo a la memoria:	$\overline{MREQ}$
Indica operación de I/O en uso:	$\overline{IORR}$
Indica que la CPU lee de la memoria:	$\overline{RD}$
Indica que la CPU escribe en memoria:	$\overline{WR}$
Señal de refresco de la memoria:	$\overline{RFSM}$
Realizado un halt de la CPU:	$\overline{HALT}$
Petición de espera:	$\overline{WAIT}$
Petición de interrupciones:	$\overline{NMI}$, $\overline{INT}$
Reset de la CPU:	$\overline{RESET}$
Líneas de control del bus:	$\overline{BUSRQ}$, $\overline{BUSAK}$

ARQUITECTURA INTERNA DEL MICROPROCESADOR ZILOG Z80



INSTRUCCIONES ASSEMBLER DEL MICROPROCESADOR ZILOG Z80

Instrucción	Código mnemónico IEEE	Código mnemónico Z80	Instrucción	Código mnemónico IEEE	Código mnemónico Z80
Instrucciones aritméticas					
Add	ADD	ADD	Compare with Increment		CPI
Add with carry	ADDC	ADC			
Subtract	SUB	SUB	Compare with Decrement		CPD
Subtract with carry	SUBC	SBC			
Increment	INC	INC	Compare multiple		CPIR
Decrement	DEC	DEC			CPDR
Compare	CMP	CP	Negative	NEG	NEG

Instrucción	Código mnemónico IEEE	Código mnemónico Z80
Instrucciones lógicas		
AND	AND	AND
OR	OR	OR
Exclusive OR	XOR	XOR
NOT	NOT	CPL
NOT carry	NOTC	CCF
Shift right	SHR	SRL
Shift left	SHL	SLA
Shift right arithmetic	SHRA	SRA
Rotate right	ROR	RRCA
		RRC
Rotate left	ROL	RLCA
		RLC
Rotate right	RORC	RR
through carry		RRA
Rotate left	ROLC	RL
Through carry		RLA
Rotate right decimal	ROR4	RRD
Rotate left decimal	ROL4	RLD
Test bit	TEST1	BIT
Instrucciones de transferencia de datos		
Load	LD	LD
Store	ST	LD
Move	MOV	LD
Block load with increment		LDI
Block load with decrement		LDD
Move block	MOVBK	LDIR
Repeat block load with decrement		LDDR
Exchange	XCH	EX
Exchange alternate register		EXX
Input	IN	IN
Input with increment		INI
Input with decrement		IND
Input block	INBK	INIR
Block input with decrement		INDR
Output	OUT	OUT
Output with increment		OUTI
Output with decrement		OUTD
Outpunt block	OUTBK	OTIR
Block output with decrement		OTDR
Sct bit	SET1	DET
Clear bit	CLR1	RES
Set carry	SETC	SCF
Set interrupt mode	SETI	IM

Instrucción	Código mnemónico IEEE	Código mnemónico Z80
Instrucciones de salto		
Branch	BR	JP
Branch if zero	BZ	JPZ
Jump relative if zero		JRZ
Branch if not zero	BNZ	JPNZ
Jump relative if not zero		JRNZ
Branch if carry	BC	JPC
Jump relative if carry		JRC
Branch if not carry	BNC	JPNC
Jump relative if no carry		JRNC
Branch if positive	BP	JPP
Branch if negative	BN	JPN
Branch if parity even	BPE	JPPE
Branch if parity odd	BPO	JPPO
Decrement and branch if not zero		DJNZ
Instrucciones de llamada a subrutinas		
Call	CALL	CALL
Restart at address		RST
Call if zero	CALLZ	CALLZ
Call if not zero	CALLNZ	CALLNZ
Call if carry	CALLC	CALLC
Call if not carry	CALLNC	CALLNC
Call if positive	CALLP	CALLP
Call if negative	CALLN	CALLN
Call if parity even	CALLPE	CALLPE
Call if parity odd	CALLPO	CALLPO
Instrucciones de retorno		
Return	RET	RET
Return if zero	RETZ	RETZ
Return if not zero	RETNZ	RETNZ
Return if carry	RETC	RETC
Return if no carry	RETNC	RETNC
Return if positive	RETP	RETP
Return if negative	RETN	RETN
Return if parity even	RETPE	RETPE
Return if parity odd	RETPO	RETPO
Return from interrupt		RETI
Return form interrupt Non-maskable		RETN
Instrucciones varias		
No operation	NOP	NOP
Push	PUSH	PUSH
Pop	POP	POP
Wait	WAIT	HALT
Adjust deminal	ADJ	DAA
Enable interrupt	EI	EI
Disable interrupt	DI	DI

El lenguaje Cobol

El procesador electrónico debe su nombre más común, ordenador o calculador, a la capacidad intrínseca de efectuar cálculos, y fue esencialmente ésta la función que realizó en la primera fase de su evolución. Sin embargo, gracias a la enorme velocidad de funcionamiento, el calculador podía ser de ayuda también en campos de aplicación diferentes al puramente matemático. Por ejemplo, podía resolver problemas relacionados con la gestión rápida de las informaciones registradas en grandes archivos, en cuya consulta, un ser humano debería haber empleado años enteros.
La exigencia de resolver rápidamente problemas específicos de un determinado ambiente por un lado, y la alta especialización necesaria para la utilización del calculador por el otro, hicieron surgir la necesidad de lenguajes de programación de nivel más elevado. Estos productos debían cumplir tres condiciones:

- **Poner a disposición del personal no especializado con el uso del calculador un lenguaje sintético y lo más cercano posible a la estructura del humano**. La finalidad de esta función es la de permitir al usuario concentrar su atención y sus esfuerzos a la búsqueda de las soluciones del problema examinado, de la misma forma en que un pintor utiliza sus colores sin conocer una sola fórmula de química.

- **Estar estructurados de modo que traten de la forma más ágil el tipo de información propio de un cierto ambiente**. Refiriéndonos específicamente al ambiente comercial, es evidente que la persona encargada del cálculo de los sueldos de los empleados de una sociedad tiene necesidad de identificar de manera sencilla la fila de carácteres alfabéticos que constituyen el nombre del empleado, y efectuar sencillos cálculos con sólo las cuatro operaciones básicas de la aritmética sobre denominaciones que componen los pagos. Es evidente que sería superfluo dotar de un lenguaje orientado a estos problemas comerciales con comandos para el cálculo de funciones de matemática superior.

- **Ser, en los límites de lo posible, independiente de la máquina utilizada para la ejecución del programa**, de manera que el software pudiese «transportarse» con pequeñas modificaciones de un calculador a otro. La transportabilidad del software, o sea la posibilidad de utilizar los mismos programas en máquinas diferentes, constituye evidentemente una garantía en el tiempo para las inversiones del usuario.

Persiguiendo estos objetivos, un grupo de fabricantes y usuarios de procesadores reunidos en la COnference of DAta SYstem Languages (CODASYL) perfiló, en 1959, las especificaciones de estandarización del COBOL (COmmon Business Oriented Language), o sea de un lenguaje de programación estudiado para la solución de los problemas que se presentan habitualmente en las actividades comerciales.
Considerando la amplitud del ambiente en que el Cobol está orientado, se comprende el motivo de su difusión y de su constante evolución como respuesta natural a las exigencias de sofisticación, facilidad de aplicación y flexibilidad sentidas por el usuario.
Por los citados motivos de difusión del lenguaje, y en consideración del tipo de aplicaciones involucradas, el texto que sigue se referirá al ANS COBOL (American National Standard COmmon Business Oriented Language); las limitaciones y la desuniformidad de este estándar se indicarán y comentarán ante las diferentes instrucciones interesadas. De acuerdo con la exigencia de proporcionar al programador un lenguaje de tipo «humano» para enseñar a la máquina a resolver un determinado problema, el elemento base del lenguaje Cobol es la palabra, análogamente a lo que sucede en cualquier idioma. En otros términos, el Cobol es un lenguaje de tipo discursivo más que de tipo simbólico, en el que se pueden mezclar, según precisas reglas sintácticas, palabras creadas por el programador y palabras, siempre autoexplicativas, propias del lenguaje. Por ejemplo, si se desea calcular la ganancia neta de una determinada operación comercial como diferencia entre el cobro y el importe bruto de la mercancía, es suficiente introducir la siguiente instrucción:



J. Pickerell/Marka

La sección de las unidades de memoria de disco en una sala de máquinas.

SUBTRACT IMPORTE-BRUTO FROM COBRO GIVING GANANCIA-NETA

donde IMPORTE-BRUTO, COBRO, GANANCIA-NETA son nombres asignados por el programador, mediante los cuales puede entenderse inmediatamente el significado lógico de la operación a realizar.
Este ejemplo da una idea de cómo el programador puede llegar a instruir un calculador «hablando» un lenguaje que sea humanamente comprensible.

Generalidades

Antes de describir las características del Cobol y la sintaxis de las instrucciones, es importante ilustrar la secuencia de los pasos lógicos y operativos que permiten obtener un programa Cobol. Es importante recordar que un calculador es un instrumento como cualquier otro, del que el hombre dispone para su propio trabajo o su propia afición. Aunque inconscientemente, cada persona pasa siempre a través de varias fases ante un problema:

1 / Análisis del problema e identificación de la solución
Supóngase que quieren unirse de forma sencilla dos trozos de madera (problema). ¿Cómo unirlos? ¿Con clavos, con tornillos o con cola? (análisis). En base a la función que debe realizar la unión y por ejemplo para la construcción de un mueble, se decide adoptar la cola (instrumento).

2 / Preparación del material
La cola debe aplicarse sobre los materiales a unir después de que se hayan preparado adecuadamente.

3 / Uso del instrumento
Cuando los materiales están preparados es posible hacer uso del instrumento elegido (en este caso la cola); terminada completamente su acción (según las especificaciones del fabricante) es posible tener el objeto terminado, respondiendo más o menos al proyecto inicial, y según la calidad de la ejecución de cada uno de los pasos considerado anteriormente.

Análogamente a lo que se ha descrito acerca del uso de la cola, el uso del calculador para la ejecución de un programa es el último de una serie de pasos que el programador debe realizar antes de obtener el resultado esperado. Abajo se han indicado todos los pasos necesarios para obtener un primer formato del programa comprensible para el procesador.

Todas las acciones esquematizadas se han efectuado fuera del procesador. El análisis del problema es una pura acción de pensamiento, mediante la cual, el analista proporciona al programador las directivas más eficaces para el trazado de un programa que pueda producir los procesos deseados.

La esquematización gráfica con la diagramación de bloques del programa también es una actividad del pensamiento, con la cual el programador presenta de forma sistética los diversos pasos lógicos que constituirán el programa. Este paso, si bien no es indispensable para el planteo del programa, es muy aconsejable, puesto que trabajando sobre una imagen visual del flujo del proceso permite evitar o corregir rápidamente errores de implantación lógica.

Para el planteo de los programas de flujo se utilizan símbolos gráficos particulares, de los que se han representado algunos ejemplos en la tabla de la pág. 954, que amplían el conjunto de los símbolos gráficos ya ilustrados.

El programa fuente

Terminada la diagramación de bloques, el programador inicia el planteo detallado del programa, o sea la traducción de los diversos pasos de proceso del diagrama en una serie de instrucciones (que respetan el formato previsto por el lenguaje elegido) escritas sobre adecuados módulos que reciben el nombre de **hojas de programación.**

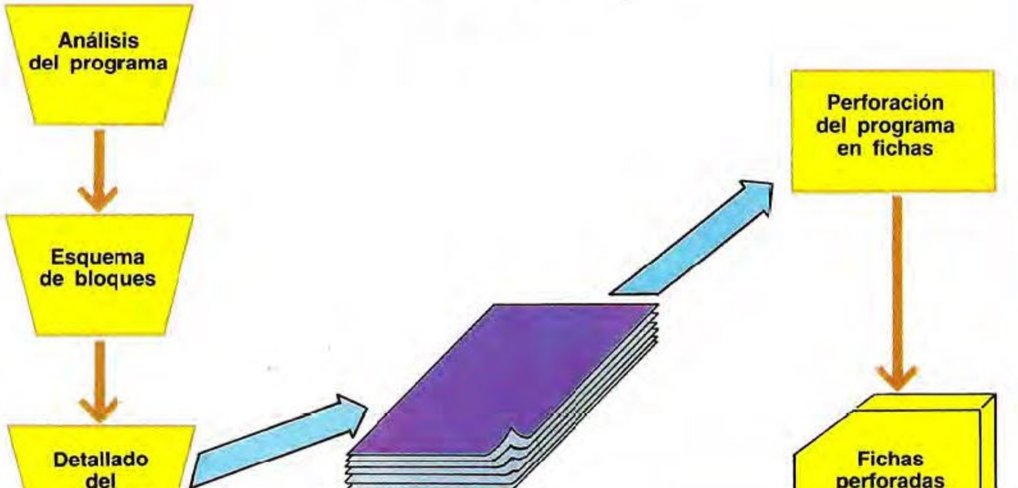
El detalle de las instrucciones según las normas que se describirán más adelante sólo es el primer paso de una secuencia de operaciones que debe aplicarse al programa antes de que pueda ser realizado por el calculador.

Al final del detalle, el programa se escribe en lenguaje simbólico sobre hojas normales de papel, que no pueden constituir una entrada válida para el calculador. Es necesario codificar las instrucciones indicadas sobre las hojas de programación en un primer formato, inteligible para la máquina, sobre un soporte adecuado: la ficha perforada.

Como puede verse en la pág. 954, una ficha está constituida por un cartón normal que puede albergar hasta 80 caracteres consecutivos.

La codificación de las instrucciones la realiza una máquina perforadora, a menudo no conectada al procesador. Cada perforadora está dotada de un teclado similar al de una máquina de escribir sobre la que se teclean, carácter por ca-

OPERACIONES FUERA DE LINEA PARA LA PREPARACION DE UN PROGRAMA



SIMBOLOGIA COBOL DE LOS DIAGRAMAS DE FLUJO

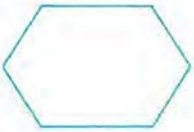
SIMBOLO — **SIGNIFICADO**



Fase inicial o final de un programa



Grupo de instrucciones utilizadas para la realización de un mismo paso lógico (procedimiento)



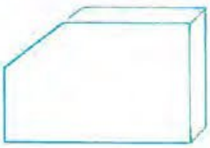
Decisión. Identifica un punto del programa en que pueden emprenderse dos acciones diferentes según se produzca un determinado suceso



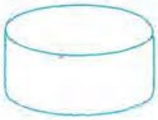
Operación de lectura o escritura en un file (en particular funciones I/O)



Ficha perforada



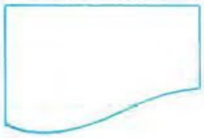
File de fichas perforadas



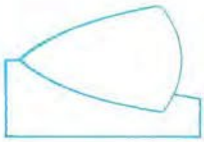
File residente en disco



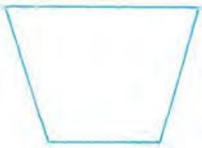
File residente en cinta magnética



File de impresión

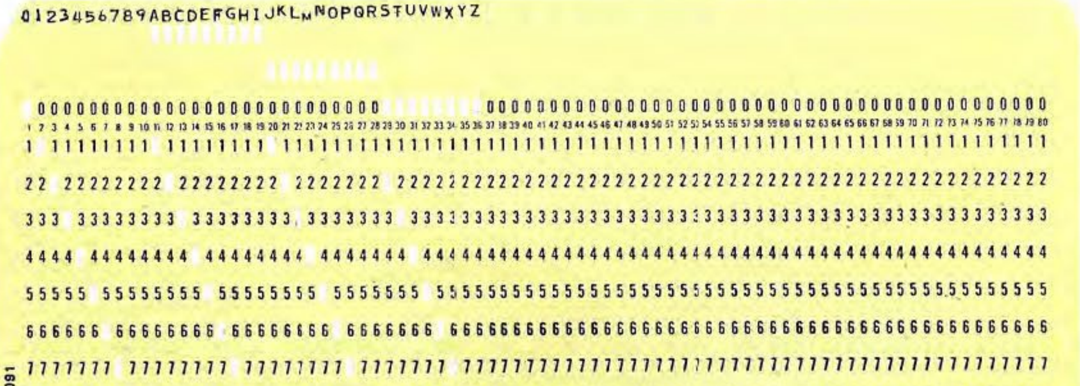


Vídeo con teclado (consola)



Operación fuera de línea realizada a velocidad humana sin ayuda de máquinas

EJEMPLO DE FICHA PERFORADA



rácter, las instrucciones Cobol indicadas sobre la hoja de programación. Cada carácter tecleado no se reproduce simplemente sobre la parte alta de la ficha, sino que está codificado con una combinación adecuada de orificios rectangulares dispuestos a lo largo de una columna.
En la tabla de abajo se ha indicado la correspondencia entre los caracteres (alfabéticos, numéricos y especiales) y su código sobre la ficha perforada según un código muy difundido (la notación 6-8 está para indicar que el carácter está codificado con dos orificios dispuestos sobre la sexta y octava línea de la ficha). La numeración de las líneas de una ficha, partiendo desde arriba, es la siguiente: 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
El paquete de fichas que se obtiene al final de la fase de digitación contiene el **programa fuente**, que sobre este soporte constituye una entrada válida para el procesador. Mediante una adecuada unidad periférica (**lector de fichas**), el calculador puede interpretar correctamente todos los caracteres simbólicos contenidos en las fichas.
A efectos lógicos, las dos operaciones son completamente equivalentes, por lo que en las figuras que siguen se indicarán con el símbolo del file todas aquellas entradas que pueden ser asimiladas por un conjunto de fichas perforadas. En otras palabras, programas fuente e informaciones simbólicas con un máximo de 80 columnas, tanto si residen en disco o se teclean directamente en consola, se indicarán con dicho símbolo. Todas las fases escritas hasta ahora permiten pasar de la búsqueda de la solución de un problema a la escritura de un programa que permita su aplicación, desde el detalle de las instrucciones a la introducción de las mismas en el calculador. Las instrucciones del programa, en este punto, todavía están expresadas en forma simbólica, es decir, en la forma prevista por la sintaxis del lenguaje. En otras palabras, si el programador ha detallado la instrucción

MOVE A TO B

existirá en la máquina una combinación de caracteres idénticos a la introducida.
Como el calculador no puede comprender el significado de la frase MOVE A TO B es necesario someter el programa a otras dos operaciones sucesivas.
Estas operaciones tendrán la finalidad de traducir la palabra MOVE, en el caso del ejemplo, en una serie de instrucciones escritas en el lenguaje (binario) propio de la máquina, para que el calculador pueda interpretarlas y realizarlas correctamente, con el fin de obtener la transferencia (MOVE) pedida por el comando.
Las operaciones indicadas son la **compilación** y el **encadenado** (linking), y deben realizarse en el orden indicado.

Compilación y encadenado

Un Compilador, cualquiera que sea el lenguaje de programación usado, es un programa de sistema que puede interpretar los comandos simbólicos contenidos en el programa fuente y producir una serie de comandos en el lenguaje pro-

CODIGO DE PERFORACION 026

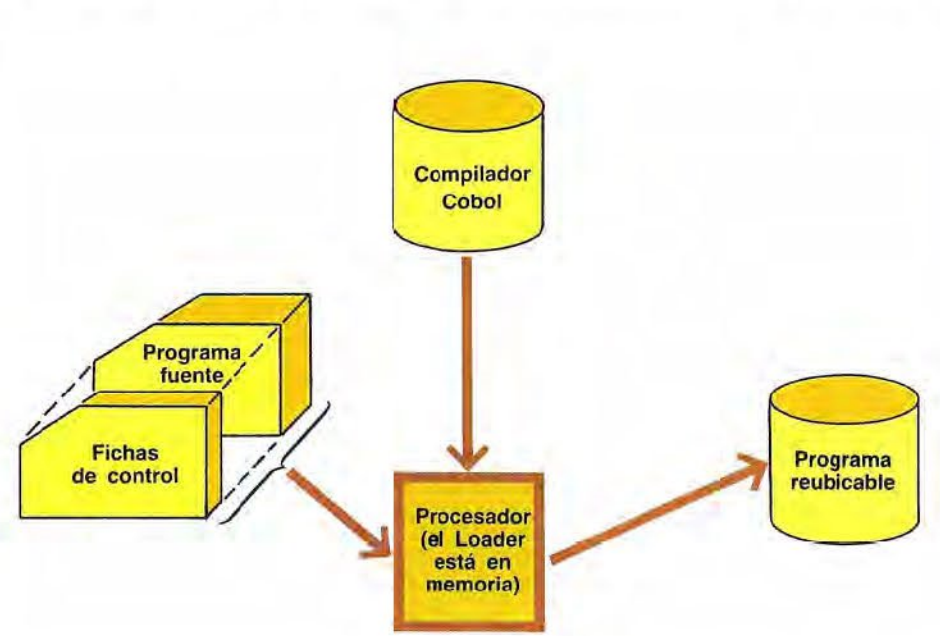
Símbolo	Código	Símbolo	Código	Símbolo	Código	Símbolo	Código	Símbolo	Código
A	12-1	P	11-7	0	0	Espacio		>	6-8
B	12-2	Q	11-8	1	1	+	12	?	12-0
C	12-3	R	11-9	2	2	'	4-8	°	7-8
D	12-4	S	0-2	3	3	(	0-4-8	|	12-5-8
E	12-5	T	0-3	4	4	)	12-4-8	"	0-6-8
F	12-6	U	0-4	5	5	*	11-4-8	—	11-5-8
G	12-7	V	0-5	6	6	,	0-3-8	#	12-7-8
H	12-8	W	0-6	7	7	—	11	$	11-3-8
I	12-9	X	0-7	8	8	.	12-3-8	%	0-5-8
J	11-1	Y	0-8	9	9	/	0-1	&	2-8
K	11-2	Z	0-9			:	5-8	b	11-7-8
L	11-3					;	11-6-8	!	11-0
M	11-4					<	12-6-8		
N	11-5					=	3-8		
O	11-6								

pio de la máquina usada, que constituyen el **programa reubicable**. Del Compilador ya se ha hablado en la exposición del lenguaje Basic.
Las acciones a realizar para compilar un programa son análogas para todos los tipos de calculador, y se activan mediante sencillos comandos directos al Sistema Operativo, contenidos en un cierto número de **fichas de control**.
Estas fichas, cuyo formato y número varían según el calculador usado, tienen esencialmente tres funciones:

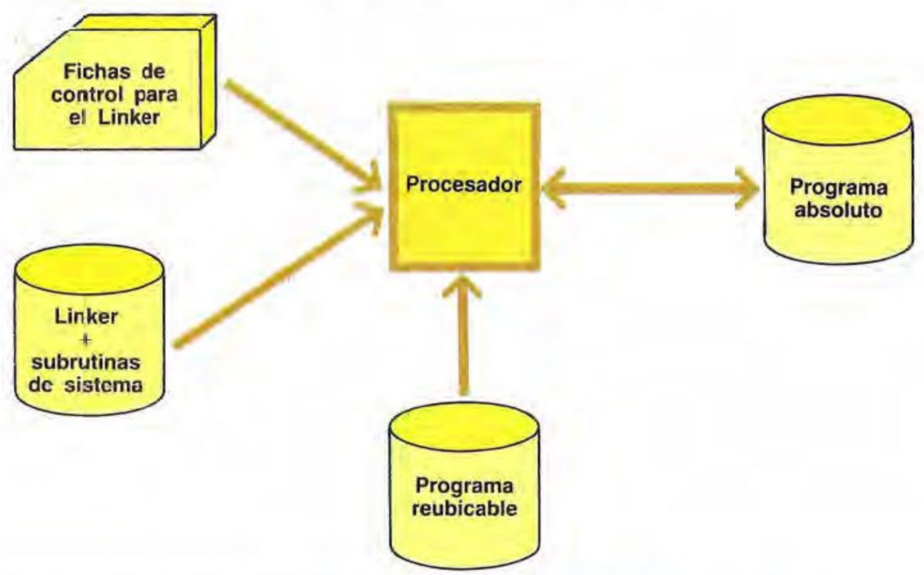
- Comunicar al sistema cuál es el Compilador que debe cargarse en memoria (Cobol, Fortran, Basic,...).
- Provocar la carga en memoria del Compilador deseado, haciendo intervenir un programa llamado Loader (cargador)
- Comunicar al Compilador el nombre del programa fuente y el nombre del file (en el caso del Cobol en disco) en que debe contenerse el producto de la compilación (programa reubicable)

La situación se indica esquemáticamente en la figura de abajo.
Al tratar las funciones del Compilador se ha dicho que éste procede a traducir las diversas instrucciones simbólicas en lenguaje máquina. La traducción se realiza utilizando sus rutinas residentes en una librería de sistema. Piénsese por ejemplo en las funciones de lectura y escritura en file: estas funciones van precedidas de las rutinas de I/O (Input/Output) genéricas que sólo se aplican a continuación al programa específico. En otras palabras, el Compilador, durante su acción, procede a crear los puntos de ataque de todas las subrutinas llamadas por las instrucciones del programa, registrando en estos puntos una serie de informaciones útiles para la siguiente operación de **encadenado**.
Además, en esta fase, de todos los datos definidos por el programador se anotan solamente las direcciones provisionales correspondientes al mapa general de la memoria. Estas direcciones (reubicables) varían en el momento en que cada subrutina se carga realmente en memoria y se ensambla, para direccionar correctamente los datos que el programa debe procesar.
El Compilador trata del modo descrito tanto las subrutinas de sistema como las escritas por el programador y residentes en otras librerías, para que puedan proporcionar el nombre mediante las adecuadas fichas de control.

ESQUEMA DE PRINCIPIO PARA LA COMPILACION DE UN PROGRAMA



ENLACE DE UN PROGRAMA OBJETO



Al final de la fase de compilación, el programa objeto reside en disco y es necesario conectarle todas las subrutinas que necesita.

Según el esquema de la figura de arriba, estas funciones las proporciona el Linker (encadenador). Este programa, así como el residente en la librería del sistema, tiene la misión de resolver todos los encadenados entre los diversos módulos reubicables.

El producto proporcionado a la salida de la fase de enlace es un **programa absoluto**, puesto que está completo en cada una de sus partes. En la figura de la pág. 958 se ha indicado una esquematización del proceso completo.

Aparte del tratamiento realizado hasta ahora sobre las operaciones a que debe someterse un programa para poderlo hacer ejecutable, es necesario indicar la siguiente precisión.

Haciendo referencia a lenguajes fuertemente estandarizados como el Cobol, si el programa fuente puede adaptarse (salvo pequeñas limitaciones) a cada procesador, el Compilador Cobol es del todo diferente de una máquina a otra, ya que cada calculador está construido según criterios tecnológicos diferentes. Es en este caso que se aprecia la estandarización del lenguaje, que permite transportar los programas de una máquina a otra compilándolos en el procesador destinado a ejecutarlos.

Después de haber descrito todos los pasos que constituyen la realización de un programa, volveremos ahora a analizar la fase de detalle de las instrucciones.

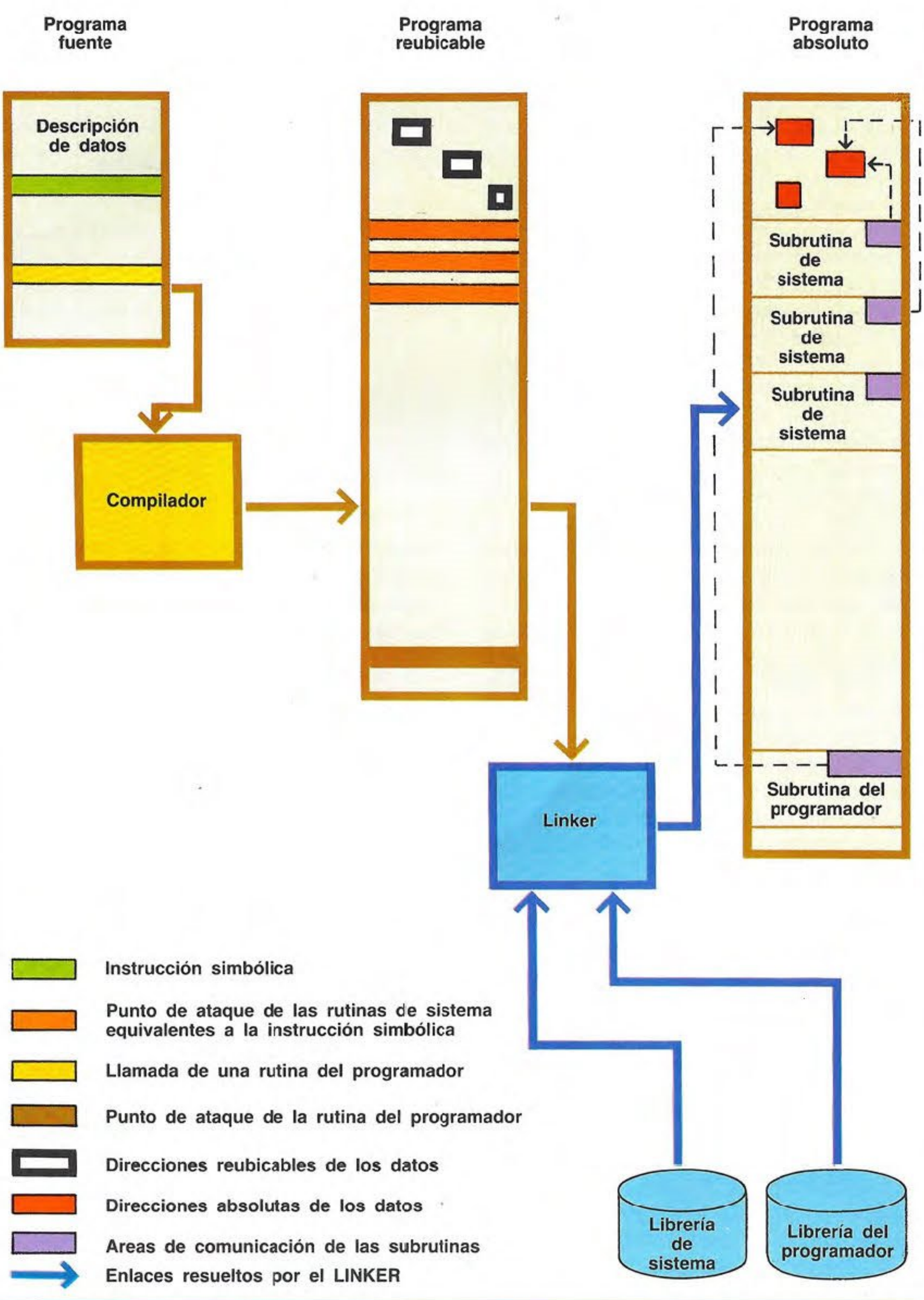
La hoja de programación Cobol y las reglas de detalle

Una hoja de programación es un módulo sobre el cual el programador escribe las instrucciones del programa; tiene la función de facilitar el trabajo de detallado, y se ha proyectado para albergar instrucciones escritas según el estándar de un determinado lenguaje.

La hoja de programación Cobol está estructurada en diferentes zonas, destinadas a contener informaciones de diferente naturaleza y significado.

Su uso no es estrictamente indispensable a los fines del correcto planteo de un programa, pero siempre es aconsejable, puesto que en ella se indican claramente algunas de las convenciones típicas del Cobol que hacen más rápido el detallado. Además, en el caso de que el programa deba perforarse sobre fichas, la utilización de la hoja de programación tiene la ventaja de hacer más rápido el trabajo de la perforación, asegurando entre amplios márgenes la correspondencia real entre el programa escrito y su codificación en fichas.

ESQUEMA DEL PROCESO DE COMPILACION Y ENLACE



HOJA DE PROGRAMACION COBOL

PROGRAMADOR ______________________ FECHA ______________

PROGRAMA ______________________ 73 76 80

COBOL

Pagina 1 3 de ______

PROGR. 4	CONT. 7	A 8	B 12	16	20	24	28	32	36	TEXTO 40	44	48	52	56	60	64	68	72
01																		
02																		
03																		
04																		
05																		
06																		
07																		
08																		
09																		
10																		
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		

En la pág. 959 se ha representado la hoja de programación. Observando la figura puede verse cómo el módulo está compuesto por dos partes principales. En la parte superior se indican algunas informaciones útiles para la documentación, mientras que la segunda parte se destina a contener palabras e instrucciones que forman parte del programa. Más exactamente, a cada línea corresponderá una ficha. Las columnas numeradas del módulo son sólo 72, mientras se ha dicho que las columnas disponibles en una ficha son 80. Efectivamente, el Compilador Cobol sólo considera que forman parte del programa los caracteres perforados de la columna 7 a la columna 72.

Es posible contraseñar cada ficha perforada de manera que se sepa con certeza a qué programa pertenece y en qué posición debe encontrarse en el interior del paquete de fichas (deck). A tal efecto se han reservado las columnas 1 a la 6 y las 37 a la 80.

Las columnas 1 a 6 pueden contener un número compuesto de la siguiente manera: las primeras tres cifras identifican el número de hoja de programación, mientras que las otras tres cifras, el número de la línea en el interior de aquella hoja. Con esto se consigue que el número completo perforado en las primeras 6 columnas de cada ficha identifique unívocamente la ficha en el interior del programa. El nombre del programa a que pertenece cada ficha puede perforarse en las columnas 73 a 80. Tanto el número progresivo de la hoja (columnas 1 a 3) como el nombre del programa se indican en la parte superior del módulo, mientras que el número de secuencia de las filas debe ser necesariamente perforado sobre la fila correspondiente (columnas 4 a 6) de la parte inferior. La numeración de las fichas según las convenciones escritas no es obligatoria, pero con cualquiera que se emplee, si las fichas no están en la secuencia correcta, el Compilador Cobol señalará el error.

Las columnas de la parte inferior de la hoja de programación, a excepción de las 4 a 6 de la que ya se ha descrito su uso, están vinculadas a reglas bien precisas de utilización.

Columna 7 — En esta columna sólo es posible perforar algunos símbolos particulares. Un trazo de unión (—) informa al Compilador Cobol que el primer carácter de la fila correspondiente es la continuación de la última palabra de la fila anterior, mientras que un asterisco (*) permite al programador utilizar la fila para insertar en la lista del programa un comentario.

Columnas 8 a 11 — Identifica un espacio de la letra A que corresponde a la columna 8; esta zona también se llama **área A** y se reserva a la perforación de los nombres de Divisiones, Secciones y Párrafos (son parte de un programa Cobol).

Columnas 12 a 72 — Delimitada por una línea más marcada e indicada con la letra B que corresponde a la columna 12, esta zona se reserva a las perforaciones de las instrucciones del programa. Si el programador tiene necesidad de insertar una o más filas entre las ya escritas, puede evitar reescribir todo el módulo utilizando las filas no numeradas y adoptando una numeración adecuada.

En la página siguiente se ha representado un ejemplo en el que se aplican todas las reglas de detallado examinadas hasta ahora, en particular el uso de la columna 7 y la inserción de otras filas utilizando la zona no numerada de la hoja. El ejemplo indicado sólo tiene finalidad educativa y resultará más claro después de la exposición de la sintaxis del lenguaje Cobol.

Estructura de un programa Cobol

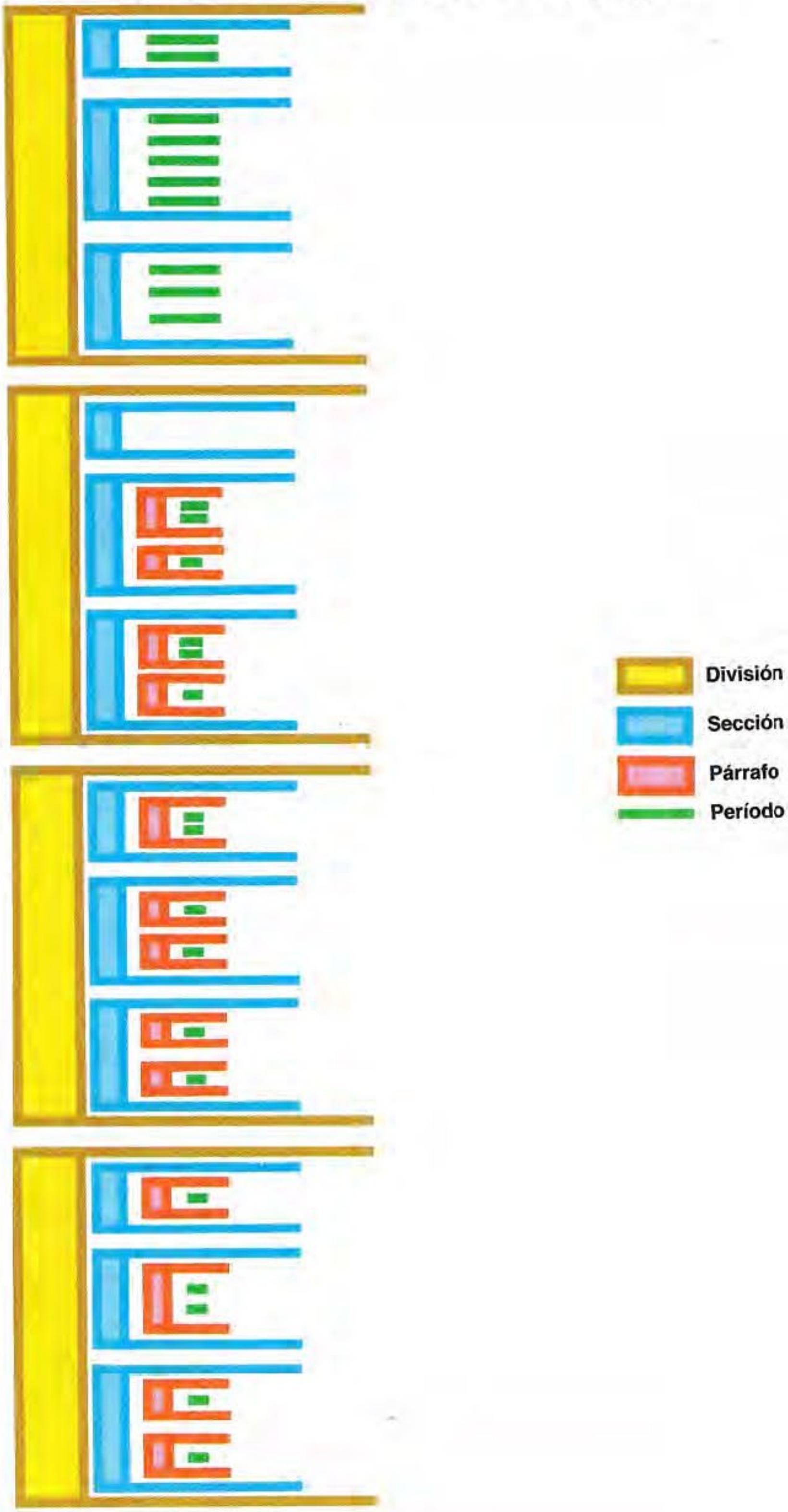
Cada programa Cobol está estructurado en divisiones (DIVISION), secciones (SECTION), párrafos y períodos, organizados jerárquicamente según el esquema de la figura de la pág. 962. Es decir, cada división está compuesta por varias secciones, cada sección por varios párrafos y cada párrafo por uno o más períodos.

PROGRAMADOR ______ FECHA ______ COBOL

PROGRAMA ______ 73 76 80 Página 1 3 de ___

```
PROGR  CONT A   B                                                         TEXTO
4      7    8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
010         PROCEDURE DIVISION.
020         PRINCIPIO SECTION.
030         ABRE-FILE.
040    *                   EL PARRAFO 'ABRE-FILE' ABRE TODOS LOS FILES
050    *                   UTILIZADOS EN EL CURSO DEL PROGRAMA.
060             OPEN INPUT FILE-ANTIGUO.
070             OPEN OUTPUT FILE-NUEVO.
080         LEE-ANTIGUO.
090             READ FILE-ANTIGUO INTO REC-ANTIGUO
100                 AT END DISPLAY 'LECTURA DEL FILE FILE-ANTIGUO ULTIMADA'
110    -        'CORRECTAMENTE' UPON PRINTER
120                 GO TO FIN.
130         PROCESA-ANTIGUO.
140             MOVE ----------
150             ADD  ----------
160         LEE-ACTUALIZACIONES.
170             READ ACTUALIZACIONES
180                 AT END ........
190
065             OPEN INPUT ACTUALIZACIONES.
155             SUBTRACT .......
```

ESTRUCTURA DE UN PROGRAMA COBOL



Un **período** es un conjunto de instrucciones (en el límite una sola) que termina con un punto. Por ejemplo, el siguiente conjunto de instrucciones es un período:

```
MOVE A TO B
MOVE C TO D
COMPUTE E = B – C.
```

Un **párrafo** está constituido por varios períodos y se identifica con un nombre elegido por el programador. El nombre puede contener un máximo de 30 caracteres, elegidos entre las letras del alfabeto inglés, los números de 0 a 9 el signo (–), utilizado con funciones de trazo de unión. El nombre del párrafo debe perforarse a partir de la columna 8 y debe cerrarse con un punto. El siguiente es un párrafo:

```
CALCULA-ITE.
    COMPUTE ITE = IMPORTE * 5/100.
    ..............................
```

Un párrafo se extiende desde el nombre hasta el principio de otro párrafo.
Varios párrafos componen una **sección**. Cada sección se identifica con un nombre que está sujeto a las mismas reglas del nombre del párrafo. Lo que permite al Compilador distinguir las dos entidades es la palabra SECTION; si se desea agrupar una serie de instrucciones como sección, la palabra SECTION debe seguir al nombre que se quiere dar a la sección. Por ejemplo, la secuencia de fichas

```
CALCULA-ITE SECTION.
CALCULA.
    COMPUTE ITE = IMPORTE * 5/100.
```

define la sección CALCULA-ITE. Cada sección debe empezar con un nombre de párrafo y termina con el principio de otra SECTION.
Es interesante precisar que, a menos que sean comandos de salto explícitos previstos por el programador, el final de un párrafo o de una sección no implica la interrupción del flujo del proceso en aquel punto. La finalidad de la subdivisión del programa Cobol en secciones y párrafos se aclarará más adelante. Mientras que la citada subdivisión en secciones y párrafos es a discrección del programador, la DIVISION es una entidad obligatoria del programa Cobol.
Un programa Cobol está constituido siempre por cuatro divisiones dispuestas en secuencia. Esta secuencia, absolutamente inalterable por parte del programador, permite al Compilador adquirir en sucesión lógica todas las informaciones necesarias para la ejecución de los comandos contenidos en el programa.
Estas divisiones, indicadas en la secuencia pedida por el Compilador, se describen a continuación.

IDENTIFICATION DIVISION. Esta división, de estructura muy sencilla, declara al Compilador el nombre del programa y permite insertar informaciones facultativas de documentación.

ENVIRONMENT DIVISION. Mediante esta división se declara al Compilador el ambiente en que el programa deberá trabajar, describiendo las relaciones que hay entre algunos componentes hardware y software del sistema. La división se compone de dos secciones. La primera, CONFIGURATION SECTION, describe el tipo y el modelo del ordenador con que se ha compilado el programa, así como el tipo y el modelo de máquina que lo realizará. La segunda, INPUT-OUTPUT SECTION, declara el nombre de los files utilizados por el programa y los soportes físicos en los que residen o residirán.

DATA DIVISION. La función de la DATA DIVISION es esencialmente la de reservar para el programa áreas de memoria principal oportunamente dimensionadas, y asociar a cada una de éstas, de manera unívoca, los nombres mnemónicos creados por el programador y utilizados en el curso de dicho programa.
Es posible subdividir a grandes rasgos las áreas de memoria reservadas en la DATA DIVISION en tres categorías.

- Una primera categoría de posiciones de memoria es la que el Compilador reserva para las operaciones de lectura y/o escritura en los files. La definición de estas áreas se realiza con la primera sección de la DATA DIVISION, o sea la **FILE SECTION** (sección file).
- En cambio, a la segunda categoría pertenecen las áreas de memoria que el programador se reserva en función de las exigencias que la lógica del programa impone y sobre las que se tiene una gestión completa. Las definiciones de estas posiciones de memoria se agrupan en la **WORKING-STORAGE SECTION** (sección área de trabajo).

- La tercera categoría de áreas en la memoria central es la de las posiciones reservadas a los intercambios de informaciones entre el programa principal y las subrutinas exteriores. El nombre de la sección en que se definen las áreas de esta última categoría es, significativamente, **LINKAGE SECTION** (sección de encadenado). Es superfluo observar que la presencia de la LINKAGE SECTION se pide únicamente en el caso en que el programa haga referencia a otros programas externos. En definitiva, la estructura completa de la DATA DIVISION es la que se indica a continuación:

DATA DIVISION.
FILE SECTION.
    Descripción de los files
WORKING-STORAGE SECTION.
    Descripción de los datos independientes
LINKAGE SECTION.
    Descripción de los datos de intercambio con rutinas externas.

PROCEDURE DIVISION. En la PROCEDURE DIVISION hay todas las instrucciones Cobol en las que el programador ha traducido el flujo lógico de las operaciones a efectuar sobre los datos. Esta división consiste en un conjunto de párrafos y secciones.
La descripción sumaria de un programa Cobol, de sus partes constituyentes y de las funciones que cada una de éstas debe asumir, ha proporcionado una idea aproximada de las peculiaridades de este lenguaje. Ahora es necesario entrar en el detalle de cada división para comprender completamente su organización, la correcta sintaxis y las reglas de detalle a respetar. En el curso de la exposición se utilizarán las convenciones de la tabla al pie de esta página.

IDENTIFICATION DIVISION

La IDENTIFICATION DIVISION es la única división compuesta sólo por párrafos.
El nombre de la división, como el de todos los párrafos componentes, debe perforarse a partir de la columna 8 (AREA A).
El formato completo de la IDENTIFICATION DIVISION se indica en la siguiente tabla.

FORMATO DE LA IDENTIFICATION DIVISION

<u>IDENTIFICATION DIVISION.</u>
<u>PROGRAM-ID.</u> nombre-programa.
[<u>AUTHOR.</u> nombre-autor.]
[<u>INSTALLATION.</u> nombre-instalación.]
[<u>DATE-WRITTEN.</u> fecha-de-escritura.]
[<u>DATE-COMPILED.</u> fecha-de-compilación.]
[<u>SECURITY.</u> tipo-de-seguridad.]
[<u>REMARKS.</u> comentarios.]

De acuerdo con las convenciones adoptadas puede observarse que el único párrafo obligatorio de esta división es el PROGRAM-ID (program identification), mientras que todos los demás son opcionales y sólo tienen funciones de documentación.
Además debe tenerse presente que, si los hay, estos párrafos deben respetar el orden indica-

CONVENCIONES ADOPTADAS EN LA EXPOSICION

Símbolo	Descripción	Convención
———	Subrayado	Identifica una entidad obligatoria en una instrucción
[ ]	Paréntesis cuadrado	Encierran una entidad, una cláusula o una serie de cláusulas adjuntables y opcionales en el formato de una instrucción
{ }	Corchete	Encierran una serie de entidades alternativas entre sí; el programador sólo puede utilizar una
	Caracteres mayúsculos	Todos los nombres reservados del Cobol están descritos en caracteres mayúsculos
	Caracteres minúsculos	Se escriben con caracteres minúsculos todos los campos o entidades a los que el programador es libre de dar el nombre o el formato que considere más oportuno
....	Puntos	Indican que la cláusula o la entidad inmediatamente anterior puede estar presente más veces en la misma instrucción

do. El párrafo REMARKS (comentarios) permite insertar un comentario bastante largo, que se considera cerrado por el primer punto que encuentra el Compilador.
Un ejemplo de IDENTIFICATION DIVISION completa es la siguiente:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJEMPLO-ID.
AUTHOR. JUAN-PEREZ-MADRID.
INSTALLATION. BARCELONA.
DATE-WRITTEN. JUNIO 84.
DATE-COMPILED. 10/06/84.
SECURITY. NINGUNA.
REMARKS. ESTE ES UN EJEMPLO SOBRE EL
         FORMATO COMPLETO DE LA
         IDENTIFICATION DIVISION DE UN
         PROGRAMA COBOL.
```

ENVIRONMENT DIVISION

En la descripción de los motivos que han conducido a la normalización de algunos lenguajes, en particular el Cobol, se ha insistido en la transportabilidad del software, o sea en la posibilidad de ejecutar un mismo programa sobre procesadores diferentes. Esta transportabilidad, salvo raras excepciones, nunca es total. Efectivamente, la normalización puede ajustar la sintaxis y la estructura de las instrucciones de manera que resulten independientes del tipo de máquina utilizada, pero no puede imponer notaciones que obliguen a los fabricantes a modificar la propia filosofía de los sistemas operativos.
En otras palabras, como el programa debe interaccionar con el sistema, debe declarar el nombre del soporte lógico que contiene los datos (file) y el del tipo de unidad física en que reside (disco, cinta, lector de fichas, etc).
Si se quieren utilizar dos máquinas gobernadas por sistemas operativos diferentes para realizar el mismo programa, será necesario no sólo recompilar el programa, sino alterar también la parte que identifica el tipo de ambiente huésped del programa, o sea la ENVIRONMENT DIVISION. A continuación se analizarán los diversos párrafos y secciones de esta división, dejando al lector la misión de proporcionar al programa los debidos parámetros de acuerdo con las especificaciones descritas en los manuales proporcionados por el fabricante del calculador utilizado en cada caso.
La secuencia de secciones y párrafos que componen la división es la siguiente:

FORMATO DE LA ENVIRONMENT DIVISION

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. nombre-ordenador.
OBJECT-COMPUTER. nombre-ordenador.
[SPECIAL-NAMES.]
[INPUT-OUTPUT SECTION.]
[FILE-CONTROL.]

Obsérvese que la ENVIRONMENT DIVISION debe estar presente en cada programa, y que los nombres de secciones y párrafos deben perforarse a partir de la columna 8, mientras que las informaciones asociadas deben empezar en la columna 12 o más adelante.

CONFIGURATION SECTION

El nombre de esta sección es extremadamente autoexplicativo; su función es la de identificar tanto el sistema operativo que compilará el programa (SOURCE-COMPUTER), como el que lo realizará (OBJECT-COMPUTER).
El nombre del ordenador que aparece en estos dos párrafos debe proporcionarse de acuerdo con las indicaciones del fabricante.

El párrafo SPECIAL-NAMES. Este párrafo, opcional, es de notable utilidad en las aplicaciones más típicas del Cobol.
Al estar orientado a problemáticas de tipo comercial, el Cobol permite imprimir un número colocándole automáticamente al lado el símbolo del dólar $.
El tema volverá a tratarse más adelante; por ahora es importante ilustrar el modo en que el Compilador Cobol puede tratar de forma análoga el símbolo de la moneda de otros países y tener así las anotaciones correctas.
La modificación es muy sencilla y se declara al Compilador en el párrafo SPECIAL-NAMES, en que se indica el carácter que debe sustituir el $ como símbolo de moneda.
Supóngase que debe tratarse con francos franceses, cuyo símbolo es F; el formato del párrafo es el siguiente:

SPECIAL-NAMES.
CURRENCY SIGN IS 'F'.

Debe observarse que el nuevo signo de moneda no debe ser ninguno de los siguientes:
— números de 0 a 9
— caracteres alfabéticos A B C D L P R S V X Z
— caracteres especiales * + – , . ; ( ) " / =

Porque el Cobol, como otros lenguajes de programación, fue realizado por primera vez en EE.UU., y es natural que prevea una representación de los números decimales según la convención utilizada en aquel país. Debe observarse que esta convención prevé el uso del punto (.) como separador entre cifras enteras y cifras decimales, mientras que un número entero se divide en grupos de tres cifras (centenas, millares, etc.) adoptando la coma.

En otras palabras, el número 4.725,8 (según la notación europea), en la convención norteamericana se escribe de la forma

4,725.8

Para permitir una representación de los números en impresión según la notación europea es suficiente insertar en el párrafo SPECIAL-NAMES la siguiente información:

DECIMAL POINT IS COMMA

que literalmente significa «el punto decimal es la coma». Las convenciones adoptadas en el párrafo SPECIAL-NAMES valen para todo el programa y no es posible alterarlas.

El párrafo FILE-CONTROL. Mediante este párrafo, el programador declara al Compilador el nombre de los diversos files usados por el programa, así como el soporte físico en que residen dichos files.

Supóngase, por ejemplo, que debe utilizarse un file que contiene datos personales residentes en disco y catalogados en el sistema con el nombre LISTIN. Si el programador ha establecido utilizar para este file en el interior del programa el nombre FILE-LIS, es necesario asociar al nombre «interno» FILE-LIS el «nombre externo LISTIN, para que el sistema pueda indicar los datos a tratar en la fase de ejecución. Esta asociación se crea en el párrafo FILE-CONTROL con la siguiente notación:

SELECT FILE-LIS
ASSIGN TO DISC LISTIN.

En general, la cláusula SELECT, que debe ser única para cada file tratado, tiene el formato indicado en la tabla de abajo.

También, para la información del nombre hardware de un file debe atenderse a las indicaciones del fabricante del sistema, por lo que el ejemplo anteriormente indicado, válido para algunos sistemas, puede no serlo para otros. Sin embargo, estas diferencias corresponden sólo a la formación del nombre-hardware.

La palabra OPTIONAL que aparece en el formato general de la SELECT declara al Compilador que el file correspondiente también puede faltar en la fase de ejecución.

Es interesante precisar que la ausencia del file en el momento de la ejecución del programa no

FORMATO DE LA CLAUSULA SELECT

SELECT [OPTIONAL] nombre-interior-file ASSIGN TO nombre-hardware

nombre-interior-file — es el nombre con que se llama el file en el interior del programa

nombre-hardware — identifica el nombre con que el sistema conoce el file y el soporte hardware en el que reside este file.

Los soportes hardware pueden ser

CARD - READER = lector de fichas
CARD - PUNCH = perforador de fichas
PRINTER = impresora
TAPE = cinta magnética
DISC o DISK = disco

FLUJO DE LOS DATOS



significa que el file no esté físicamente en el soporte hardware declarado, sino sólo que no se ha asignado al programa.
Esta asignación debe realizarse externamente al programa mediante comandos de control propios del sistema operativo utilizado.
Para aclarar las ideas considérese el caso de un programa que deba usar el file del ejemplo anterior y deba imprimir su contenido en papel. El diagrama de flujo correspondiente puede esquematizarse como en el gráfico de arriba.

La INPUT-OUTPUT SECTION del programa debe contener la selección de los files correspondientes junto con los respectivos nombres internos, o sea

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT FILE-LIS.
        ASSIGN TO DISC LISTIN.
        SELECT IMPRESION ASSIGN TO PRIN-
        TER.
```

En la fase de ejecución es necesario comunicar al sistema que el programa utilizará el file LISTIN, mientras que no es necesario asignar la impresora, puesto que ésta siempre está disponible porque está gestionada por el sistema.
Aparte de que los comandos para tales operaciones varían en función del sistema operativo, su secuencia puede sintetizarse como sigue:

1.ª ficha:

comando-de-asignación-file LISTIN.

2.ª ficha:

comando-de-ejecución-programa
nombre-programa.

DATA DIVISION

Como ya se ha dicho, la función de la DATA DIVISION es la de reservar al programa áreas de memoria principales oportunamente dimensionadas y asociar a cada una de ellas, de manera unívoca, los nombres mnemónicos creados por el programador y utilizados en el curso de dicho programa. Una primera categoría de posiciones de memoria es la que el Compilador reserva para las operaciones de lectura y/o escritura en los files. La definición de estas áreas se efectúa en la primera sección de la DATA DIVISION, la **FILE SECTION**. En cambio, a la segunda categoría pertenecen las áreas de memoria que el programador se reserva en función de las exigencias que la lógica del programa impone y de las cuales tiene una gestión completa. Las definiciones de estas posiciones de memoria están agrupadas en la **WORKING-STORAGE SECTION**.

FILE SECTION

La primera sección de la DATA DIVISION es la FILE SECTION, en la que se declaran al Compilador las características de todos los files que utiliza el programa. Por tanto, es obvio que esta sección puede omitirse en el caso en que el programa no necesite ningún file ni en la entrada ni en la salida. Para que pueda existir un programa de este tipo, prácticamente no hay ningún empleo en ambiente comercial, donde los procesos están dedicados típicamente a la manipulación de archivos.
Cada file se describe al Compilador mediante el indicador de nivel, el nombre y una serie de cláusulas, algunas de ellas facultativas, que especifican sus características.
El indicador de nivel está constituido por la palabra reservada FD, abreviación de File Descrip-

tion, perforada a partir de la columna 8 del período de descripción. Por ejemplo:

DATA DIVISION.
FILE SECTION.
FD DESCRIPCION.

En el ejemplo, DESCRIPCION es el nombre del file utilizado en el interior del programa. El nombre del file debe perforarse siempre a partir del margen B, y mejor a partir de la columna 12.

Descripción de los datos: la cláusula BLOCK CONTAINS. Antes de analizar la continuación del período de descripción del file es interesante indicar algunas informaciones relativas a los files secuenciales.
Un file secuencial en disco puede leerse tomando un record cada vez. Este modo de proceder, sobre todo si los records a leer son muchos y tienen longitudes muy pequeñas, puede prolongar mucho los tiempos de proceso. Efectivamente, cada lectura o escritura en disco comporta operaciones mecánicas como el posicionamiento de la cabeza sobre la pista o el sector oportuno, cuyos tiempos de ejecución son como promedio mil veces superiores a los típicos de las operaciones efectuadas en la memoria.
Por tanto, es preferible organizar los records de un file en bloques, de manera que cada operación de lectura o escritura física en disco trate simultáneamente más records. En otras palabras, el Cobol transfiere siempre un bloque cada vez, desde o hacia el file. Si no se declaran cláusulas relativas al «bloqueo» de los records, el Compilador reserva a estas operaciones un área de memoria igual a la longitud del record y, por tanto, el sistema quedará limitado a realizar tantos accesos en el disco como cuantos records leídos o escritos haya en el file. Si, en cambio, el programador declara el bloque compuesto por un cierto número n de records, el Compilador reservará un área (buffer) adecuada para contener enteramente n records.
De este modo, el acceso al disco para la lectura de un nuevo bloque sólo se realizará cuando el programa haya tratado todos los n records del bloque anterior. El procedimiento se ha esquematizado en el gráfico de la página siguiente, en el que se ha supuesto que deben leerse y procesarse los records de un file secuencial con bloqueo = 3. En este caso, la descripción del file debe contener la cláusula

BLOCK CONTAINS 3 RECORDS

La forma general de las cláusulas se indica en la tabla del final de la página.
Son cláusulas válidas las siguientes:

BLOCK CONTAINS 10 RECORDS
BLOCK CONTAINS 1 TO 10 RECORDS
BLOCK CONTAINS 100 CHARACTERS
BLOCK CONTAINS 10 TO 900 CHARACTERS

Debe observarse que, si se usa la declaración CHARACTERS, el número de caracteres declarados debe contener también los caracteres de control.
La cláusula BLOCK puede omitirse, y esto equivale a escribir

BLOCK CONTAINS 1 RECORDS

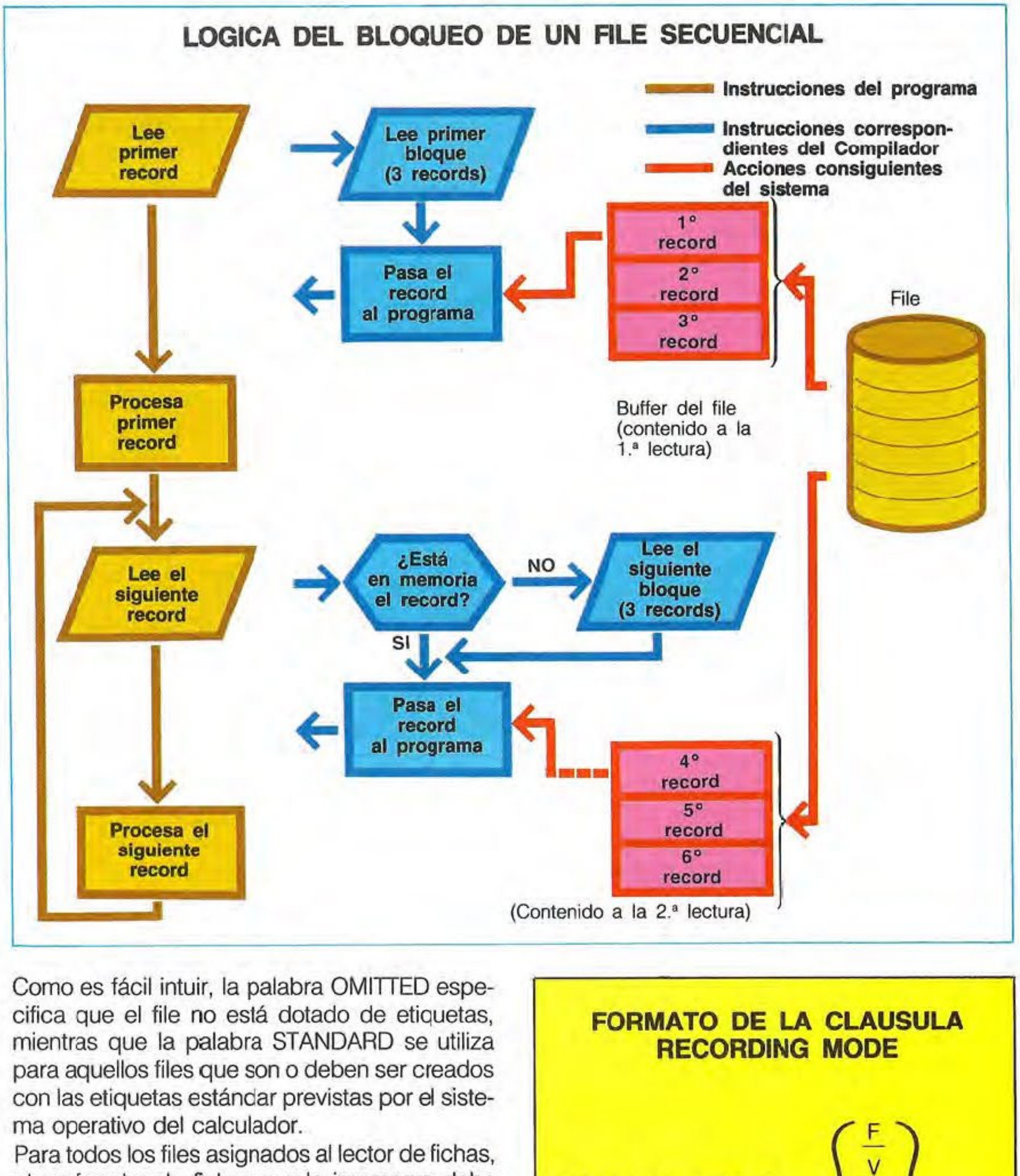
La cláusula LABEL RECORD. Esta cláusula **siempre debe estar presente en la descripción de un file** y tiene la función de especificar si todo el file descrito tiene o deberá tener etiquetas (labels).
El formato general de la cláusula es el siguiente

FORMATO DE LA CLAUSULA LABEL RECORDS

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORD}}\ \text{IS} \\ \\ \underline{\text{RECORD}}\ \text{ARE} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{OMITTED}} \\ \\ \underline{\text{STANDARD}} \end{array} \right\}$$

FORMATO DE LA CLAUSULA BLOCK CONTAINS

$$\underline{\text{BLOCK}}\ \text{CONTAINS}\ [\text{entero-1}\ \underline{\text{TO}}]\ \text{entero-2} \left\{ \begin{array}{l} \underline{\text{CHARACTERS}} \\ \underline{\text{RECORDS}} \end{array} \right\}$$



Como es fácil intuir, la palabra OMITTED especifica que el file no está dotado de etiquetas, mientras que la palabra STANDARD se utiliza para aquellos files que son o deben ser creados con las etiquetas estándar previstas por el sistema operativo del calculador.
Para todos los files asignados al lector de fichas, al perforador de fichas o a la impresora debe utilizarse la cláusula

LABEL RECORD IS OMITTED

La cláusula RECORDING MODE. A menudo, los records contenidos en un file tienen todos la misma longitud. Sin embargo, por exigencias específicas es posible tratar otros tipos particulares de memorización de los records.
El formato general de la cláusula es el siguiente:

FORMATO DE LA CLAUSULA RECORDING MODE

RECORDING MODE IS { F | V | U | S }

Omitir la cláusula equivale a escribir RECORDING MODE IS F, y el Compilador supone que los records del file tienen una longitud fija (Fixed). Esta longitud se calcula automáticamente en base a la descripción del record, como se verá más adelante.

El significado de las letras que aparecen en la cláusula es el siguiente

V Es la inicial de Variable y declara explícitamente que los records del file tienen longitudes variables. El modo en que el Compilador adquiere el campo de variabilidad de estas longitudes quedará claro cuando se hable de la descripción del record. Sin embargo, debe tenerse presente que, durante la escritura de un file con RECORDING MODE V, el sistema asocia a cada record un campo que contiene la longitud del record, y a cada bloque otro campo en el que se indica la longitud del bloque. Estos campos son completamente gestionados por el sistema operativo y no es necesario tenerlos en cuenta en la descripción del record. A continuación se indica un ejemplo de descripción de un file con records de longitud variable:

```
FD ARCHIVO
   LABEL RECORD IS STANDARD
   RECORDING MODE IS V
   RECORD CONTAINS
   1 TO 200 CHARACTERS
   BLOCK CONTAINS
   1 TO 45 RECORDS.
```

La cláusula RECORD CONTAINS todavía no se ha descrito, pero es de interpretación inmediata.

U La letra U, inicial de la palabra inglesa Undefined (indefinido), caracteriza un file en el que los records pueden tener, entre un intervalo precisado, longitudes cualesquiera. En este caso, el reconocimiento del tipo de record tratado lo efectúa el programador mediante un criterio cualquiera. Es importante observar que en el caso RECORDING MODE U debe omitirse la cláusula BLOCK, como en el siguiente ejemplo:

```
FD FILE-U
   LABEL RECORD IS STANDARD
   RECORDING MODE IS U
   RECORD CONTAINS
   100 TO 200 CHARACTERS.
```

S La cláusula RECORDING MODE IS S especifica un file en el que los records lógicos tienen una longitud (en caracteres) que supera las dimensiones del bloque. En este caso, los records se «fragmentan» (spanned, en inglés) en más de un bloque. Un file con RECORDING MODE S puede contener tantos records de longitud fija como records de longitud variable; esta información la conoce el Compilador en base a la correspondiente descripción del record y/o de la cláusula RECORD CONTAINS. En el siguiente ejemplo

```
FD FILE-S
   LABEL RECORD IS STANDARD
   RECORDING MODE IS S
   RECORD CONTAINS
   1 TO 500 CHARACTERS
   BLOCK CONTAINS
   1 TO 300 CHARACTERS.
```

El file FILE-S tiene records de longitud variable y, como puede observarse, la máxima longitud del record es superior a la máxima dimensión del bloque. Obsérvese que en el caso de RECORDING MODE S, el valor del bloqueo debe proporcionarse en CHARACTERS y no en RECORDS.

La cláusula RECORD CONTAINS. Especifica explícitamente la longitud del record. Ésta también puede ser asumida por el Compilador sobre la base de las descripciones del record o de los records contenidos en el file.
Su formato completo se indica en la pág. 971.

Estructura de los records. Durante el análisis de las diversas cláusulas para la descripción del file se ha dicho varias veces que el Compilador puede obtener de las descripciones del record sus dimensiones en caracteres. Es interesante recordar que el Compilador debe reservar en la memoria tantas posiciones como sean necesarias para contener el record. Efectivamente, ésta es el área en que queda disponible para el programa un record procedente del disco, o en que el programa compone el record antes de la operación de escritura en disco.
Un record está compuesto por una serie de caracteres agrupados lógicamente en campos. Mientras una operación sobre un file, por ejemplo de lectura, transfiere en memoria un record entero, el programa tiene la posibilidad de direccionar y tratar el record tanto en su totalidad

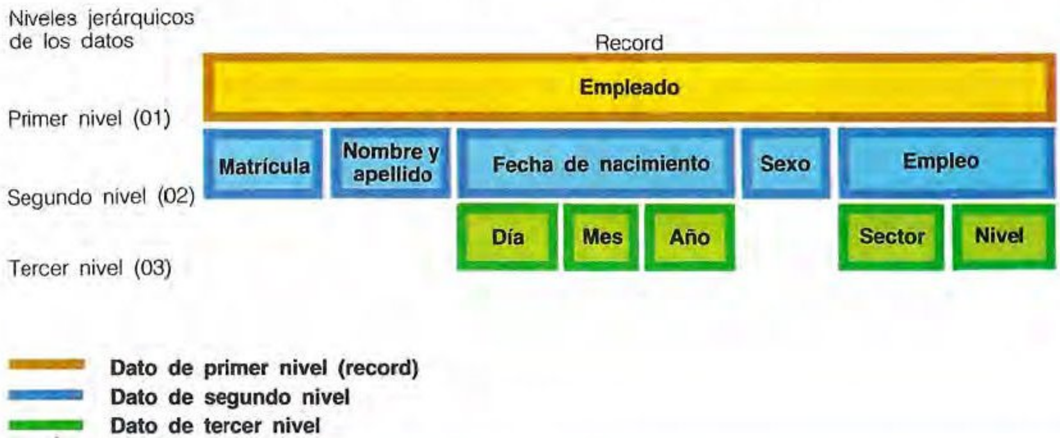
FORMATO DE LA CLAUSULA RECORD CONTAINS

RECORD CONTAINS [entero-1 TO] entero-2 CHARACTERS

entero-1 — indica el número mínimo de caracteres que puede constituir ur record en el caso de files con el record de longitud variable, de tipo indefinido o de tipo «spanned».

entero-2 — indica el número máximo de caracteres de un record.

DEFINICION Y ESTRUCTURA DE UN RECORD



como en sus campos componentes, siempre que éstos se hayan definido oportunamente. Por ejemplo, el file EMPLEADOS contiene records de longitud fija de 100 caracteres.
La estructura del record es la esquematizada arriba.
Este esquema establece un orden jerárquico que permite direccionar y modificar totalmente o en parte el área de memoria que alberga el record EMPLEADO. A esta jerarquía hay asociados números de nivel como puede verse en la descripción indicada en el listado de la parte superior de la pág. 972.
La descripción de cada campo utiliza una fila o bien una ficha. A cada campo hay asociado, al principio de la ficha, un número de dos cifras (número de nivel) comprendido entre 01 y 49, que identifica el nivel jerárquico de aquel campo con respecto al campo anterior. El nivel 01 identifica todo el record. Esto significa que, por ejemplo, modificar el contenido del campo EMPLEADO equivale a modificar de una sola vez los contenidos de todos los campos que tienen números de nivel mayores.
Y viceversa, también es posible modificar el contenido del campo AÑO-NACIMIENTO (nivel 03), sin que se altere ningún otro campo del record EMPLEADO.
No es necesario que los números de nivel sean progresivos, aunque es aconsejable asignar a campos contiguos números de nivel al menos con paso 5, reservándose de esta manera la posibilidad de definir con ulteriores detalles uno cualquiera de los campos ya descritos (ver listado de abajo de la pág. 972).
La palabra PIC que aparece en algunas de las filas de los ejemplos es la abreviación de la palabra reservada PICTURE e identifica, mediante la notación que le sigue, la longitud y el tipo del campo. El tipo de carácter que el campo puede albergar se declara con el símbolo que sigue inmediatamente a la palabra PIC:

DESCRIPCION DE LOS RECORDS DEL FILE EMPLEADOS (1)

```
FD   EMPLEADOS
     LABEL RECORD STANDARD
     RECORDING MODE IS F
     RECORD CONTAINS 100 CHARACTERS.
01   EMPLEADO.
     02  MATRICULA                    PIC 999.
     02  NOMBRE-APELLIDO              PIC X(30).
     02  FECHA-NACIMIENTO.
         03  DIA-NACIMIENTO           PIC 99.
         03  MES-NACIMIENTO           PIC 99.
         03  AÑO-NACIMIENTO           PIC 99.
     02  SEXO                         PIC A.
     02  EMPLEO.
         03  SECTOR-EMP               PIC 9(3).
         03  NIVEL-EMP                PIC 9(3).
```

DESCRIPCION DE LOS RECORDS DEL FILE EMPLEADOS (2)

```
FD   EMPLEADOS
     LABEL RECORD STANDARD
     RECORDING MODE IS F.

01   EMPLEADO.
     05  MATRICULA                    PIC 9(3).
     05  NOMBRE-APELLIDO              PIC X(30).
     05  FECHA-NACIMIENTO
         10  DIA-NACIMIENTO           PIC 9(2).
         10  MES-NACIMIENTO           PIC 9(2).
         10  AÑO-NACIMIENTO           PIC 9(2).
     05  SEXO                         PIC A.
     05  EMPLEO.
         10  SECTOR-EMP               PIC 9(3).
         10  NIVEL-EMP                PIC 9(3).
     05  FILLER                       PIC X(54).
```

X = caracteres alfanuméricos, tanto letras, números como espacios en blanco.

A = caracteres alfabéticos (letras de A a Z y espacios en blanco)

9 = caracteres numéricos (cifras de 0 a 9)

Considérese, por ejemplo, la descripción del campo NOMBRE-APELLIDO: la notación PIC X(30) declara al Compilador que el área de memoria reservada a aquel campo debe contener un máximo de 30 caracteres alfanuméricos.
Obsérvese que escribir 05 MATRICULA PIC 9(3) o bien 05 MATRICULA PIC 999 es completamente equivalente a efectos del Compilador, pero el segundo formato es menos práctico, especialmente si el campo es muy largo.
En el caso de campos subdefinidos, como por ejemplo FECHA-NACIMIENTO, el Compilador no necesita la definición de las dimensiones y del tipo del campo, puesto que puede calcular su longitud como suma de las longitudes de los campos componentes, y además asume por definición el campo del nivel superior como alfanumérico. O sea, el campo FECHA-NACIMIENTO tiene evidentemente una longitud total de 6 caracteres, pero al estar compuesto por campos numéricos [PIC 9(2)] se considera globalmente como campo alfanumérico. Más adelante se volverá sobre este tema.
A pesar de las aparentes diferencias, los ejemplos mostrados son completamente equivalentes. En el primer caso, por haber declarado expresamente la cláusula RECORD CONTAINS 100 CHARACTERS, el Compilador reserva al record un área de 100 caracteres a pesar de que se hayan descrito campos del record para un total de 46 caracteres. En el record entonces hay un campo de 54 caracteres que, al no ha-

ber sido descrito, no puede utilizarse. En el segundo ejemplo se obtiene el mismo resultado insertando en la descripción del record el campo de 54 caracteres que tiene por nombre la palabra reservada FILLER (relleno).
Efectivamente, un campo definido FILLER ocupa posiciones de memoria, pero no es referenciable por el programa. Así, el Compilador, al efectuar las sumas de las longitudes sencillas, reservará automáticamente 100 caracteres, y esto justifica la omisión de la cláusula RECORD CONTAINS 100 CHARACTERS.
Finalmente, obsérvese que en el segundo ejemplo se ha omitido la cláusula RECORDING MODE F, puesto que es superflua. Efectivamente, si el file hubiese sido de longitud variable, habría necesitado o la cláusula RECORD CONTAINS número-mínimo TO número-máximo CHARACTERS, o la definición, a nivel 01, de otro record de longitud diferente al ya definido.

WORKING-STORAGE SECTION

La WORKING-STORAGE SECTION es la sección de la DATA DIVISION en que se describen todos los datos y las áreas de trabajo utilizados en el curso del programa. Estas áreas se destinan a contener valores fijos o valores simultáneos generados en el flujo del proceso.
Es interesante observar explícitamente que la mayor parte de las reglas que se examinarán a continuación, relativas a la descripción de los campos en WORKING-STORAGE, son aplicables a la descripción de los records de la FILE SECTION. En la descripción de aquella sección se ha querido evitar deliberadamente proporcionar nociones no indispensables al tratamiento: su uso en aquella parte, si bien previsto por el Cobol y por tanto sintácticamente correcto, habría distraído la atención del lector sobre la estructura lógica de las secciones. En consecuencia, de ahora en adelante, todas las reglas escritas deben entenderse como válidas también para las descripciones del record en la FILE SECTION salvo indicación en contra.

El nivel 77. Como ya se habrá observado, a la descripción de un campo-datos hay asociado siempre un número de nivel que identifica su posición jerárquica en el interior de un área de memoria.
Un campo se llama **elemental** si no puede descomponerse posteriormente en otros campos de nivel más profundo. Considérese por ejemplo el campo FECHA-PROCESO:

```
01 FECHA-PROCESO.
   05   DIA        PIC 9(2).
   05   MES        PIC 9(2).
   05   AÑO        PIC 9(2).
```

Aquí, todos los campos de nivel 05 son campos elementales.
Supóngase que debe definirse un campo elemental, por ejemplo PROVINCIA, que no pertenezca a ningún otro campo y que a su vez no sea subdefinido. En este caso es posible utilizar dos tipos de definición:

```
01   PROVINCIA   PIC X(2).
77   PROVINCIA   PIC X(2).
```

Las dos definiciones son completamente equivalentes y sólo difieren por el número de nivel. La primera utiliza el nivel 01, y puede escribirse en cualquier punto de la WORKING-STORAGE SECTION, mientras que la segunda, al utilizar el número de nivel 77, debe escribirse al principio de la sección, como se indica en los dos ejemplos que siguen:

- **primer caso**

```
..............
..............
WORKING-STORAGE SECTION.
..............
..............
01 CAMPO-1.
   05 SUBCAMPO-1       PIC X(30).
   05 SUBCAMPO-2.
   10 SUBCAMPO-21      PIC 9(3).
   10 SUBCAMPO-22      PIC 9(3).
..............
..............
01 PROVINCIA         PIC X(2).
```

- **segundo caso** (nivel 77)

```
..............
..............
WORKING-STORAGE SECTION.
77 PROVINCIA PIC X(2).
..............
..............
```

El número de nivel 77 no puede utilizarse para la definición de un record en la FILE-SECTION.

El nivel 88. Como el campo PROVINCIA es un campo alfanumérico de dos caracteres [PIC X(2)], puede contener, por ejemplo, la sigla de la matrícula automovilística de dicha provincia. Si en el curso del programa se tiene necesidad de verificar si la provincia examinada es la de Barcelona, deberá escribirse (a partir de la columna 12)

```
IF PROVINCIA IS EQUAL TO 'B'
```

Esta verificación puede hacerse de forma diferente si el campo PROVINCIA está definido, por ejemplo, de la siguiente manera:

```
01  PROVINCIA        PIC X(2).
    88 BARCELONA     VALUE 'B'.
    88 MADRID        VALUE 'M'.
    88 VALENCIA      VALUE 'V'.
    88 ............  VALUE '...'.
    88 ............
```

En este caso, la anterior verificación puede escribirse del modo

```
IF BARCELONA
..............................
```

En otras palabras, el nivel 88 permite asociar a un campo elemental, no necesariamente al nivel 01 o 77, un nombre condicional.
Otro ejemplo se muestra en el listado de abajo. En este caso, para saber si el empleado examinado es un hombre o una mujer, puede escribirse IF HOMBRE... en lugar de IF SEXO IS EQUAL TO 'V'.

El nivel 66 y la cláusula RENAMES. El último número de nivel especial utilizado en el Cobol es el nivel 66. Su empleo, está ligado exclusivamente a la cláusula RENAMES, que permite referenciar un dato elemental o un grupo de datos ya descritos con un nombre diferente al utilizado en la descripción. Considérese, por ejemplo, el record EMPLEADO definido así:

```
01  EMPLEADO.
    05 MATRICULA              PIC 9(3).
    05 NOMBRE-APELLIDO.
       10 APELLIDO            PIC X(20).
       10 NOMBRE              PIC X(10).
    05 FECHA-NACIMIENTO.
       10 DIA-NACIMIENTO      PIC 9(2).
       10 MES-NACIMIENTO      PIC 9(2).
       10 AÑO-NACIMIENTO      PIC 9(2).
    05 SEXO                   PIC A.
```

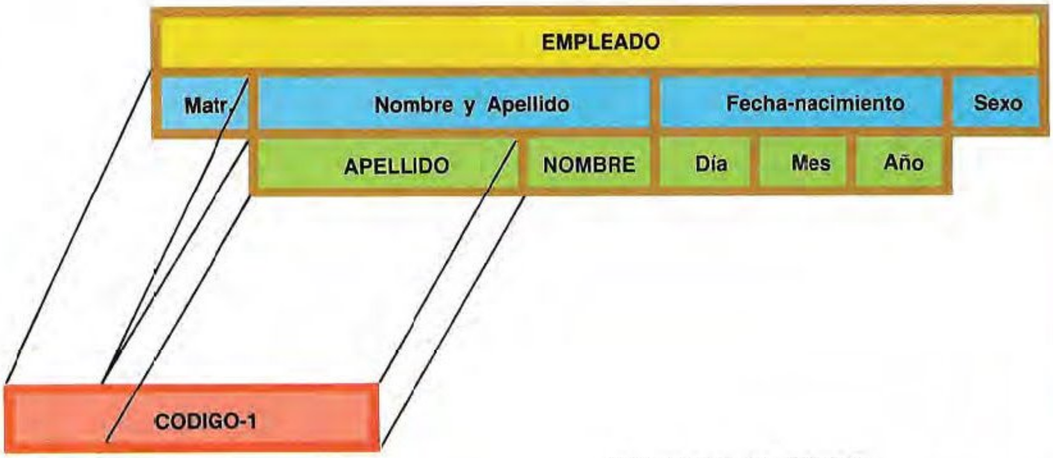
Supongamos ahora que debe reagruparse en un campo único de nombre CODIGO-1 el número de matrícula (MATRICULA) y sólo el apellido del empleado (APELLIDO) sin efectuar operaciones de movimiento dentro del programa. En este caso, el reagrupamiento se obtiene mediante la cláusula RENAMES, o bien escribiendo lo siguiente:

```
01  EMPLEADO.
    05 MATRICULA              PIC 9(3).
    05 NOMBRE-APELLIDO.
       10 APELLIDO            PIC X(20).
       10 NOMBRE              PIC X(10).
    05 FECHA-NACIMIENTO.
       10 DIA-NACIMIENTO      PIC 9(2).
       10 MES-NACIMIENTO      PIC 9(2).
       10 AÑO-NACIMIENTO      PIC 9(2).
    05 SEXO                   PIC A.
66  CODIGO-1 RENAMES MATRICULA THRU
    APELLIDO.
```

EMPLEO DEL NIVEL 88

```
01   EMPLEADO.
     05   MATRICULA                          PIC 9(3).
     05   NOMBRE-APELLIDO                    PIC X(30).
     05   FECHA-NACIMIENTO.
          10   DIA-NACIMIENTO                PIC 9(2).
          10   MES-NACIMIENTO                PIC 9(2).
          10   AÑO-NACIMIENTO                PIC 9(2).
     05   SEXO                               PIC A.
          88   HOMBRE      VALUE    'V'.
          88   MUJER       VALUE    'H'.
     05   EMPLEO.
          10   SECTOR-EMP                    PIC 9(3).
          10   NIVEL-EMP                     PIC 9(3).
     05   FILLER                             PIC X(54).
```

ESQUEMA DE LAS AREAS DE MEMORIA SOMETIDAS A LA CLAUSULA RENAMES



Ocupaciones de memoria:
Código-1: 33 caracteres
Empleado: 40 caracteres

De esta manera se comunica al Compilador que reserve un área de memoria de nombre CODIGO-1 alfanumérica y con longitud total de 33 caracteres, o bien para contener simultáneamente los campos del record que van desde MATRICULA a (THRU) APELLIDO inclusive (ver gráfico de arriba).
La cláusula RENAMES también puede utilizarse simplemente para referenciar un dato con un nombre diferente al que se ha descrito previamente. Por ejemplo:

66 NACIMIENTO RENAMES FECHA-NACIMIENTO.

En definitiva, el formato general de la cláusula RENAMES es el indicado en la tabla de abajo. El significado de nombre-dato-1, nombre-dato-2 y nombre-dato-3 se ha descrito en los ejemplos indicados anteriormente.

La cláusula REDEFINES. La exigencia de referir un grupo de datos contiguo de forma diferente a la permitida por la descripción original puede sentirse muy frecuentemente en el ámbito de las programaciones Cobol. La cláusula RENAMES, descrita anteriormente, es seguramente el método menos usado para resolver el problema, puesto que presenta el inconveniente de ocupar otro espacio de memoria para la disposición del dato con el nuevo nombre. En cambio, la cláusula REDEFINES permite redefinir el espacio reservado a un cierto dato sin ocupar otros espacios de memoria.
Sin embargo, debe subrayarse que la utilización de la cláusula REDEFINES sólo es posible cuando se está completamente seguro de que las dos definiciones deberán utilizarse en tiempos diferentes. En otras palabras, si por ejemplo el programa utiliza en un determinado momento el área DATA PR (proceso de datos) y después el área ART (artículo), puede evitarse definir separadamente las dos áreas en la WORKING-STORAGE SECTION (ocupando por tanto dos áreas de memoria) recurriendo a la siguiente escritura:

FORMATO DE LA CLAUSULA RENAMES

66 nombre-dato-1 RENAMES nombre-dato-2 [THRU nombre-dato-3]

```
01 DATA-PR.
   05 DIA                      PIC 9(2).
   05 MES                      PIC 9(2).
   05 AÑO                      PIC 9(2).
01 ART REDEFINES DATA-PR.
   05 SERIE                    PIC X.
   05 NUMERO                   PIC 9(3).
   05 FILLER                   PIC X(2).
```

Como la cláusula REDEFINES necesita que el área refinidora y la refinida tengan la misma longitud, en el ejemplo se ha insertado un campo FILLER de dos caracteres en el área ART.

Un mismo campo puede redefinirse varias veces según las modalidades ya descritas: sin embargo debe tenerse presente que las cláusulas REDEFINES siguientes siempre deben hacer referencia al primer campo de la cadena. Suponiendo que se utilice todavía el campo DATA-PR y que se quiere redefinir posteriormente como campo NUM (número de protocolo) es correcto escribir lo siguiente:

```
01 DATA-PR.
   05 DIA                      PIC 9(2).
   05 MES                      PIC 9(2).
   05 AÑO                      PIC 9(2).
01 ART REDEFINES DATA-PR.
   05 SERIE                    PIC X.
   05 NUMERO                   PIC 9(3).
   05 FILLER                   PIC X(2).
01 NUM REDEFINES DATA-PR       PIC 9(6).
```

La cláusula REDEFINES no puede aplicarse a los períodos de descripción de record de la FILE SECTION, ni a campos con número de nivel 66 y 88. La cláusula debe ir seguida inmediatamente por la descripción del campo al que se hace referencia y debe tener el mismo número de nivel. Como se verá más adelante, el Compilador puede implantar el contenido de un campo al valor establecido por el programador mediante la cláusula VALUE. Considérese por ejemplo el campo ESTAB (establecimiento) a definir como sigue:

```
01 STAB.
   05 DOMICILIO         PIC X(2) VALUE 'B'.
   05 NUM-STAB          PIC 9(4).
```

En estas condiciones no es posible utilizar STAB como redefinición de otro campo. En otras palabras, es erróneo escribir por ejemplo:

```
01 DATA-PR.
   05 DIA               PIC 9(2).
   05 MES               PIC 9(2).
   05 AÑO               PIC 9(2).
01 STAB REDEFINES DATA-PR.
   05 DOMICILIO         PIC X(2) VALUE 'B'.
   05 NUME-STAB         PIC 9(4).
```

Es interesante precisar que la cláusula VALUE, usada para la declaración de un nombre condicional al nivel 88, no impide el uso de la cláusula REDEFINES. Es decir, el ejemplo que sigue es correcto:

```
01 DATA-PR.
   05 DIA               PIC 9(2).
   05 MES               PIC 9(2).
   05 AÑO               PIC 9(2).
01 STAB REDEFINES DATA-PR.
   05 DOMICILIO         PIC X(2).
      88 BARCELONA      VALUE 'B'.
      88 MADRID         VALUE 'M'.
   05 NUM-STAB          PIC 9(4).
```

Debe indicarse que las notaciones

```
   05 DOMICILIO         PIC X(2) VALUE 'B'.
```

y

```
   05 DOMICILIO         PIC X(2).
      88 BARCELONA      VALUE 'B'.
```

no son equivalentes, puesto que con la primera se impone al Compilador implantar en el área DOMICILIO el valor B, mientras que con la segunda el programador se reserva la gestión del contenido de DOMICILIO pidiendo al Compilador que sólo asocie el nombre BARCELONA al grupo de caracteres B para poder verificar su existencia en el campo DOMICILIO. El uso del nivel 88 se ha descrito anteriormente.

La cláusula PICTURE. La cláusula PICTURE (PIC), ya introducida y usada en los ejemplos anteriores, proporciona informaciones al Compilador acerca del número y del tipo de caracteres contenidos en un dato elemental. Además, con esta cláusula, el programador puede indicar al Compilador toda una serie de operaciones a efectuar sobre caracteres del campo. Estas operaciones, orientadas sobre todo a los campos a utilizar en la fase de impresión, permi-



Monitorización computerizada de una planta química.

ten insertar símbolos particulares y cancelar o enmascarar caracteres de dicho campo.
El formato general de la cláusula PICTURE se indica en la siguiente tabla y debe especificarse

> **FORMATO DE LA CLAUSULA PICTURE**
>
> [{ PICTURE | PIC } IS símbolos.]

para todos los campos elementales, tanto a nivel 77 como a cualquier otro nivel comprendido entre 01 y 49.
La cláusula no se llama para los campos compuestos, puesto que el Compilador puede calcular las dimensiones del campo compuesto sumando las dimensiones de los campos componentes y suponiendo por definición el campo compuesto como alfanumérico, independientemente de las clases de los datos descritos.

Es decir, el campo

```
01 FECHA-NACIMIENTO.
   05 DIA                PIC 9(2).
   05 MES                PIC 9(2).
   05 AÑO                PIC 9(2).
```

es interpretado, en cuanto a referido globalmente, como campo alfanumérico de 6 caracteres.
Es importante subrayar que la cláusula PICTURE tiene una importancia fundamental para la veracidad de los resultados de un proceso. Efectivamente, un programa formal y sintácticamente correcto puede proporcionar resultados inalcanzables o conducir a terminaciones anómalas sólo porque contiene una definición errónea de un campo. Esta cláusula es muy importante, y sus implicaciones se esclarecerán oportunamente más tarde. Antes de continuar observemos que a continuación siempre se adoptará el formato abreviado

PIC símbolos.

El conjunto de los símbolos utilizables en la

cláusula PICTURE es el siguiente:

A X 9 P X * $ B O + – . , / S V CR DB

Los símbolos S V CR DB sólo pueden aparecer una vez en el ámbito de una misma PICTURE, mientras que la coma puede repetirse, aunque no inmediatamente, después de otra coma.
La PICTURE puede contener un máximo de 30 símbolos, o bien es posible escribir

01 NOMBRE PIC X(35).

mientras que no se permite la forma

01 NOMBRE PIC XXX...(35 símbolos X) ...XXX.

puesto que los símbolos que siguen a la palabra PIC son 35.
La sintaxis y el significado de la cláusula son diferentes para los campos numéricos, alfabéticos y alfanuméricos.

Descripción de los campos numéricos. Un campo destinado a acoger exclusivamente valores numéricos está descrito en la WORKING-STORAGE SECTION con un símbolo de PICTURE constituido por una combinación de los caracteres 9 V P S.
El símbolo 9 identifica una cifra (en base 10) del conjunto que constituye el campo. El contenido de los campos numéricos se entiende siempre en el sistema decimal, también si su representación interna se realiza según la codificación declarada con la cláusula USAGE, de la que se hablará más adelante.
Por ejemplo, el descrito por la línea

01 CAMPO-1 PIC 9 (10).

es un campo que puede albergar, como máximo, un número entero constituido por 10 cifras. Por definición, un campo numérico no puede contener caracteres que no sean cifras. Por tanto, en base a esta definición, un campo numérico no puede contener ni el punto decimal. Para poder indicar la posición de este separador se recurre al símbolo V, que identifica la posición virtual. Es decir, el símbolo V no ocupa espacio en memoria, pero establece una referencia para el Compilador de manera que permita una correcta alineación de los datos tratados en aquel campo. El campo numérico descrito por la línea

01 CAMPO-2 PIC 9(3)V9(3).

ocupa en memoria 6 caracteres y puede albergar números reales de tres cifras enteras y tres decimales como máximo. Si al CAMPO-2 se le asigna, por ejemplo, el valor 123456, este número será recibido y tratado como 123.456.
El carácter P permite extender el número contenido en el campo con tantos ceros (0) como cuantos símbolos P hay presentes en el símbolo de la PICTURE. Suponiendo que el dato a transferir es 8741 y que la definición del campo es

01 CAMPO-3 PIC 9(4)9(5)V.

el número se recibirá y tratará como

874100000.

mientras, si la definición es

01 CAMPO-3 PIC VP(5)9(4).

el número será tratado como

.000008741

El caracter S, si se emplea, debe ser el primero en la combinación de los símbolos de la PICTURE y permite tratar el valor numérico contenido en el campo junto con su signo algebraico. Si en el símbolo de la PICTURE se omite el carácter S, el contenido del campo se considera siempre positivo. Por ejemplo, el campo descrito por la línea

01 CAMPO-4 PIC S9(8).

puede albergar, tanto el valor +425, como el valor –425. Adoptando el símbolo S se tiene además la posibilidad de presentar el signo algebraico en impresión.
Es interesante subrayar la importancia que asume la correcta definición de un campo numérico. Efectivamente, si bien un campo definido alfanumérico (PIC X) puede albergar también cifras, éstas se tratarán exclusivamente como cualquier otro carácter no numérico.
En otras palabras, si el campo está definido como numérico (PIC 9) es posible tratar su contenido realmente como un número.
Por tanto, el uso de campos numéricos no sólo permite poder tratar su contenido mediante las

operaciones aritméticas, sino también presentarlos correctamente en impresión con la inserción física del punto decimal y del signo algebraico anexo.
Por tanto, la definición de un campo establece, además de su extensión en memoria, también su código según el tipo de caracteres que deba albergar.

Descripción de los campos alfabéticos

Por **campo alfabético** se entiende un campo que puede albergar todos y sólo los caracteres del alfabeto inglés de la A a la Z, más el espacio en blanco (blank), indicado con ƀ.
Para indicar al Compilador posiciones de memoria de tipo alfabético, deberán insertarse en la descripción tantas A como cuantos caracteres compongan el campo.
Supóngase que se desea definir el campo NOMBRE-APELLIDO alfabético de 30 caracteres; su descripción será

01 NOMBRE-APELLIDO PIC A(30).

El campo así definido puede acoger por ejemplo el dato:

JUAN JOSE PEREZ

pero no el dato:

PROF. JUAN JOSE PEREZ

puesto que entre los caracteres de esta última cadena hay comprendido el punto.
Considérese ahora el caso en que un programa tome un campo de 8 caracteres alfabéticos, de los cuales los dos primeros identifican el modelo, los siguientes tres la serie y los últimos tres el mes de producción de un determinado artículo. Si, por exigencias de impresión, los grupos de caracteres citados deben estar separados por uno o más espacios en blanco, puede recurrirse al símbolo B de la cláusula PICTURE:

01 CODIGO PIC A(2)BBA(3)BA(3).

Es decir, suponiendo que el código leído sea AZXYWSET, el contenido de CODIGO será

AZ XYW SET

Por tanto, el programador puede obtener, en determinadas posiciones del campo, la inserción de tantos espacios en blanco como cuantas B hayan en la cláusula PICTURE.

Descripción de los campos alfanuméricos

Un **campo alfanumérico** puede albergar una cadena de caracteres compuesta por letras, números, caracteres especiales y espacios en blanco en cualquier orden.
El símbolo que declara al Compilador esta clase de datos es la letra X. El Compilador considera también alfanumérico un campo en que estén presentes al mismo tiempo los símbolos de definición de al menos dos clases de caracteres. Es decir, todos los campos descritos en el ejemplo siguiente son alfanuméricos:

01 SERIE-ARTICULO	PIC AA9(2).
01 PRODUCTOR	PIC X(30).
01 DESCRIPCION	PIC AAAXXXX.
01 DESCRIPCION	PIC A(3)C(4).

También los campos alfanuméricos prevén una serie de símbolos llamados de edición, que insertados oportunamente en la cláusula PICTURE, permiten algunas operaciones particulares sobre los caracteres.
Considérese el caso en que el campo

01 SERIE-ARTICULO PIC AA9(2).

deba presentarse en impresión en la forma Serie/Número. En este caso es suficiente definir el campo de impresión como sigue:

05 SERIE-IMPRIME PIC A(2)/9(2).

Por tanto, si el dato contenido en SERIE-ARTICULO fuese SR12, una vez desplazado en SERIE-IMPRIME asumiría la forma:

SR/12

El símbolo 0 utilizado en una PICTURE de tipo alfanumérico permite insertar, con la misma lógica ya vista para los símbolos B y /, uno o más caracteres 0 en la posición especificada en la cláusula. Otros símbolos, orientados a la predisposición en la impresora para los campos numéricos, se analizan a continuación.

* Sustituye en impresión un cero inicial que se encuentra en la posición ocupada por el símbolo en la PICTURE.

Z — Sustituye en impresión tantos ceros iniciales como cuantas Z hay especificadas con espacios en blanco.

, — Inserta una coma en la posición especificada en la PICTURE, siempre que el carácter que la precede no se haya suprimido.

$ — Tiene un doble uso:

a) supresión de ceros no significativos y sustitución del último cero suprimido con el símbolo de moneda.
b) Inserción del símbolo de moneda en la posición especificada.

En el caso a), en la PICTURE deberán haber presentes tantos símbolos $ como cuantos sean los ceros que se desean suprimir. En el caso b) debe especificarse un solo carácter $.

+ – — Pueden utilizarse, según la misma lógica del símbolo $, como caracteres de inserción o de supresión.

CR DB — El sufijo CR es la abreviación de CREDIT, mientras que DB es la abreviatura de DEBIT. Sólo pueden aparecer como

Símbolo	Dato a imprimir	Picture del campo de impresión	Valor impreso
*	0000	* * * * *	* * * * * *
	0000	* * * *	* * * *
	1234	* * * *	1234
	0012	* * * *	* * 12
	0012	* * * 9	* * 12
	0012	* 9(3)	* 012
Z	0000	ZZZZ	
	1230	ZZZZ	1230
	0012	ZZZZ	12
	0012	ZZZ9	12
	0012	Z9(3)	012
,	12345	99,999	12,345
	00123	ZZ,ZZZ	123
	00123	* * , * * *	* * * 123
.	12345	9(3).9(2)	123.45
$	12345	$$$$.99	$123.45
	1234	9(4)	$1234
	0000	$Z(3)9	$ 0
	0000	$Z(4)	
+ y –	15	– 9(2)	15
	– 15	– 9(2)	– 15
	– 15	9(2) –	15 –
	00	– 9(2)	00
	15	+ 9(2)	+ 15
	– 15	+ 9(2)	– 15
	15	9(2) +	15 +
	123	– – 9(2)	123
	– 012	– – 9(2)	– 12
	000	– – – –	
	012	+ + 9(2)	+ 12
	– 001	+ + + 9	– 1
	000	+ + + +	
	123	+ + 99	+ 123
CR y DB	78912	$$$$.99CR	$789.12
	– 00478	$$$$.9(2)CR	$4.78CR
	– 00478	$$$$.9(2)DB	$4.78DB

último símbolo de una PICTURE y permiten imprimir, a la derecha del número, el sufijo CR y el DB (si el número es negativo). En caso contrario, se imprimen dos espacios en blanco.

Para aclarar los conceptos expuestos en la tabla de la página anterior, delante de cada símbolo se han indicado ejemplos. Para cada uno de ellos se ha indicado

- el valor numérico a transferir en impresión
- la PICTURE del campo de impresión destinado a albergarlo
- el correspondiente resultado sobre el papel.

Para tener una panorámica completa correspondiente a los tipos de campo sobre los que pueden operar las instrucciones que serán analizadas a continuación, en la tabla de abajo se han indicado todos los símbolos utilizables de acuerdo con tres clases de datos que el Cobol gestiona.

PROCEDURE DIVISION

La exposición realizada hasta ahora ha ilustrado las funciones y las reglas sintácticas y de detalle de las tres primeras divisiones de un programa Cobol. A pesar de que la PROCEDURE DIVISION, cuarta y última división prevista por el Cobol, sea la única en que el programador traduce el flujo lógico de las operaciones a efectuar sobre los datos, no debe identificarse con el propio programa. Es decir, el programador debe analizar el problema y aplicar las eventuales soluciones metodológicas con una visión de conjunto que comprenda las cuatro divisiones del programa.

Las instrucciones que constituyen la PROCEDURE DIVISION pueden organizarse según tres niveles jerárquicos decrecientes.

El nivel más bajo de grupo lo constituyen la **frase** o el **período**, o bien una serie de instrucciones realizables en secuencia, o bien una sola instrucción. Un agrupamiento de este tipo se encuentra, por ejemplo, en los puntos del programa en que deben seguirse caminos diferentes según se verifique o no una determinada condición. Esta fase de decisión se sintetiza en el Cobol, como en la mayor parte de los lenguajes de programación, con la instrucción IF, cuyo caso será ilustrado ampliamente más adelante. Para ilustrar el significado de la palabra «frase», considérese el caso de la pág. 982, donde quiere calcularse el cociente entre la va-

SIMBOLOS PARA LA DESCRIPCION DE LOS DATOS (PICTURE)

Clase del dato	Definición de clase			Símbolos operativos			Símbolos predisposición impresión (EDITING)											
	X	A	9	S	V	P	*	Z	0	B	,	.	$	+	—	DB	CR	/
ALFANUMERICO	SI	SI*	SI*	NO	NO	NO	NO	NO	SI	SI	NO	NO	NO	NO	NO	NO	NO	SI
ALFABETICO	NO	SI	NO	NO	NO	NO	NO	NO	NO	SI	NO	NO	NO	NO	NO	NO	NO	NO
NUMERICO	NO	NO	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI

Carácter admitido (verde)

Carácter no permitido (rojo)

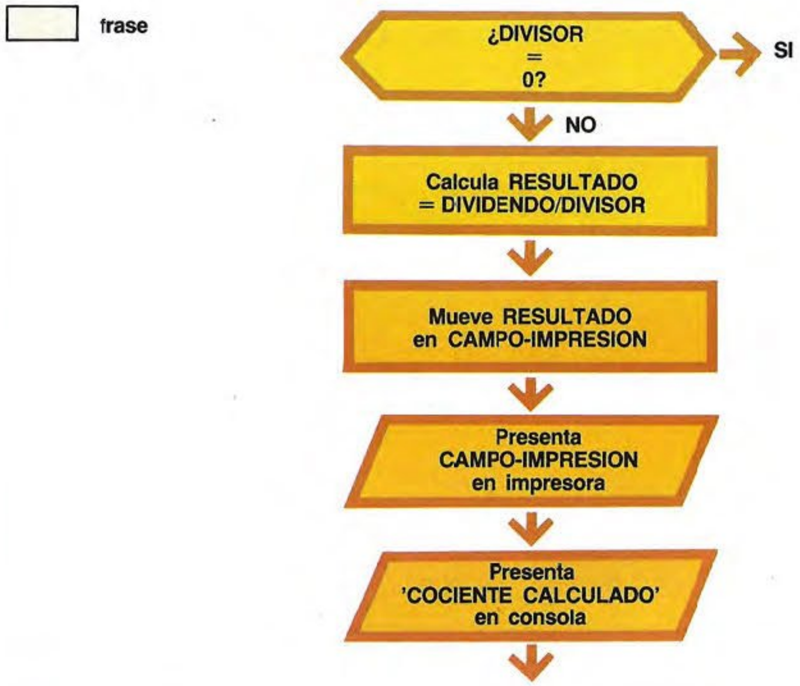
* Para definir un dato alfanumérico, junto a éste debe haber por lo menos otro símbolo de definición de clase (A, X, 9)

riable DIVIDENDO y la variable DIVISOR. Naturalmente, este cociente sólo puede calcularse si DIVISOR es diferente de 0. Así pues, en el caso en que se verifique esto, se desea

1 / calcular el resultado
2 / desplazar el resultado en un campo destinado a la impresión
3 / presentar este campo en la impresora
4 / imprimir el mensaje 'COCIENTE CALCULADO' en la consola del sistema.

En este caso, las cuatro operaciones corresponden a cuatro instrucciones Cobol diferentes y constituyen una frase, puesto que deben ser realizadas en secuencia lógica como respuesta al producirse un suceso fijado. Con el fin de hacer familiares al lector las instrucciones del lenguaje, en la ilustración de abajo se ha indicado también el correcto detallado de la parte de PROCEDURE DIVISION correspondiente al esquema. Debe observarse que una frase siempre se cierra con un punto.
Un programa Cobol, como cualquier otro programa, está constituido por una serie de instrucciones dispuestas en secuencia. Esta secuencia puede estar subdividida lógica y estructuralmente en un cierto número de **párrafos,** o bien

```
IF   DIVISOR NOT EQUAL TO 0
     COMPUTE RESULTADO = DIVIDENDO / DIVISOR
     MOVE RESULTADO TO CAMPO-IMPRESION
     DISPLAY CAMPO-IMPRIME UPON PRINTER
     DISPLAY '** COCIENTE CALCULADO = ' RESULTADO
             UPON CONSOLE.
```

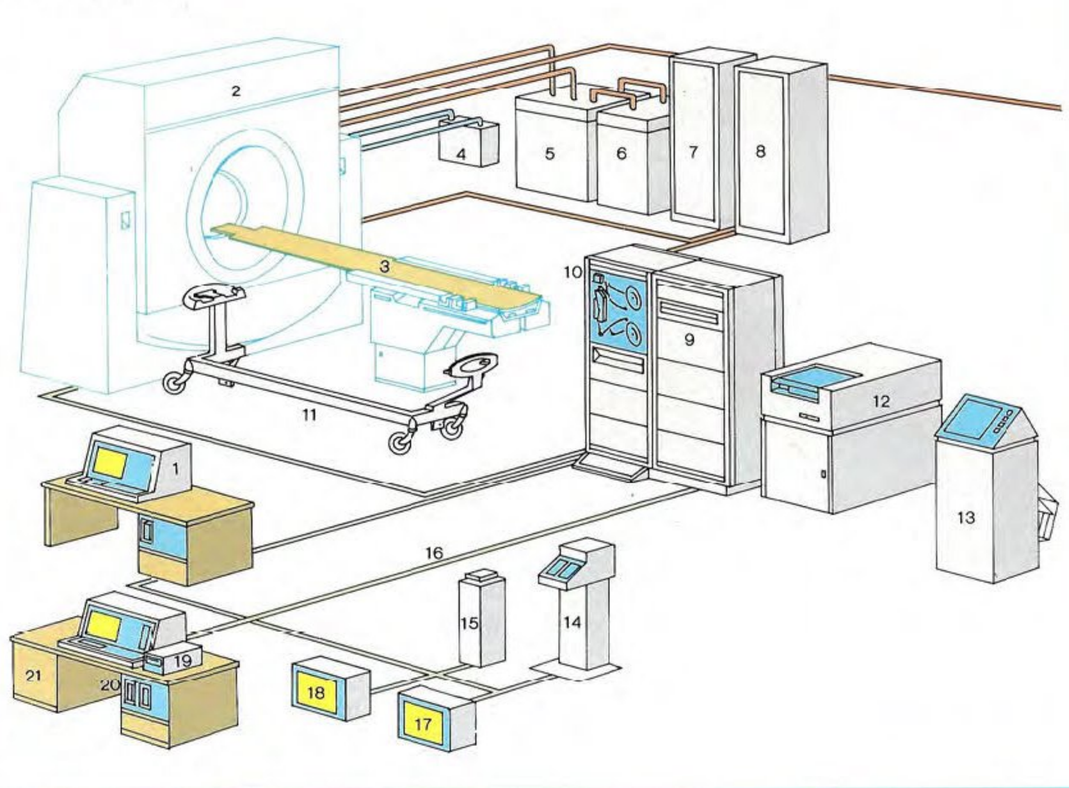


El diagnóstico computerizado (2)

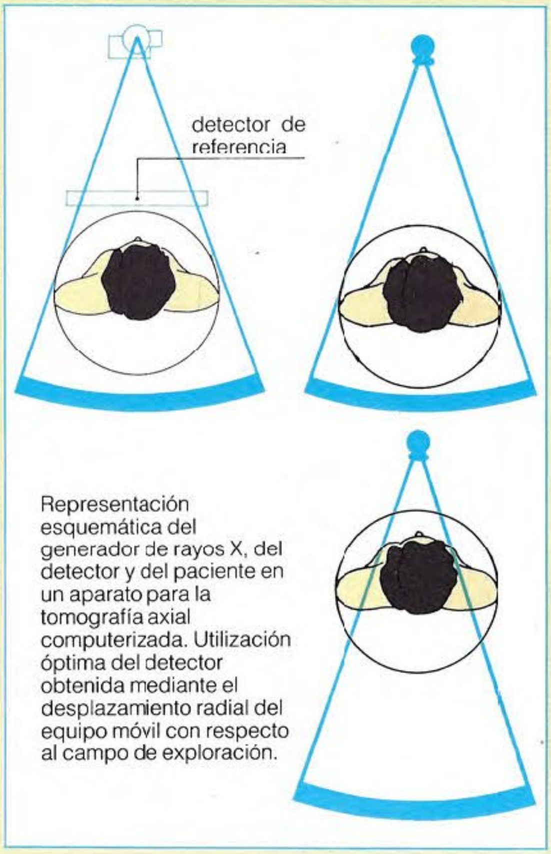
La técnica tomográfica es con mucho la que más se ha beneficiado con la intervención en el campo biomédico. Los primeros aparatos tomográficos aparecieron a principios de los años setenta y fueron aceptados inmediatamente como un suceso revolucionario en la historia de la investigación biomédica. Por primera vez fue posible examinar con detalle una sección axial cualquiera del cuerpo de un paciente mediante imágenes con unas características de calidad y de resolución netamente superiores a las de las radiografías ordinarias proporcionadas por las técnicas tomográficas tradicionales.

La tomografía axial computerizada (TAC) se basa en la reconstrucción con ordenador de una imagen global a partir de diversas imágenes radiográficas tomadas en diversas posiciones alrededor del cuerpo del paciente. Un generador de rayos X de alta resolución se posiciona a lo largo de una circunferencia, orientado hacia el centro. En el punto de la circunferencia diametralmente opuesto se posiciona, a lo largo de un arco de unos 40° (correspondiente a la anchura del haz X emitido), una matriz de detectores. Cuando el cuerpo del paciente se interpone entre la fuente y el detector, la amplitud de la señal proporcionada en la salida de cada uno de los sectores que componen este último es proporcional a la absorción selectiva realizada por el sector del cuerpo subtendido por el detector. La salida digitalizada producida por la matriz de detectores después de la recepción de un impulso X se memoriza y, después, se hace girar un cierto ángulo el conjunto generador/detector y se procede a la memorización de otra imagen. Cuando el sistema ha descrito un arco de 360 °, procesando numéricamente con el ordenador las imágenes memorizadas, es posible obtener una imagen de la sección del cuerpo explorada.

Representación esquemática de conjunto de los diversos componentes de un sistema *Tomoscan* (de 1 a 9 sistema base): 1. Consolas y monitor del operador; 2. Explorador con tubo de radiación y matriz de detección; 3. Soporte para el paciente; 4. Unidad de enfriamiento con agua; 5. Armario del tetrodo; 6. Transformador de alta tensión; 7. Cuadro de potencia; 8. Cuadro central; 9. Ordenador con microprocesador dedicado para la reconstrucción de la imagen; (de 10 a 21 opcionales): 10. Unidad de cinta; 11. Carro; 12. Disco de gran capacidad; 13. Plotter-Impresora electrostático; 14. Cámara fotográfica multiforme; 15. Fotocámara polaroid; 16. Línea de enlace de datos; 17. Monitor en blanco y negro; 18. Monitor en color; 19. Unidad para cassettes magnéticas; 20. Segunda unidad para disco flexible (disponible en la consola del operador y en la consola remota); 21. Consola remota.



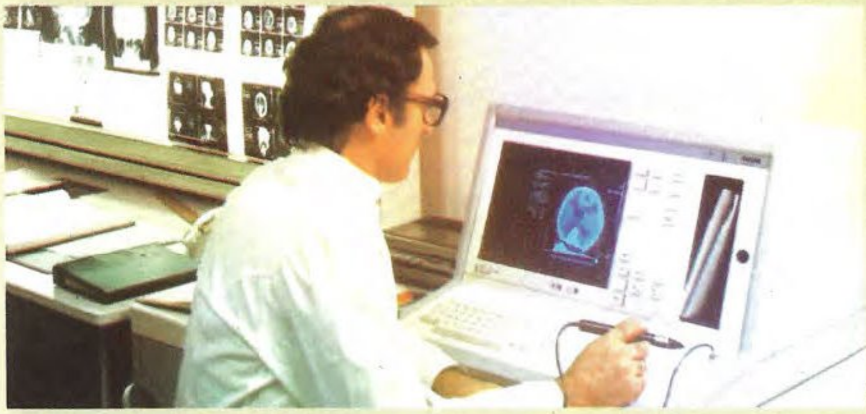
Un aparato tomográfico típico es el Tomoscan, *representado en la pág. 983. El generador de rayos X trabaja a tensiones de 100 a 120 kV y emite impulsos de 1 a 2 ms de duración. Según la resolución que se desea para la imagen, el número de impulsos emitidos durante una exploración completa de 300 ° puede variar de 350 a 1200. La matriz de detección, constituida por unos 600 detectores englobados en una cámara única presurizada que contiene xenón a 20 atmósferas, tiene un arco de cobertura de 43,5 °. El tiempo necesario para producir una imagen de alta definición es de unos 21 s (3 a 10 s son necesarios para completar la rotación de 160 °), pero existe la posibilidad de tener una visualización preliminar de la imagen, procesada en base a los registros correspondientes a la primera mitad de la exploración, después de 1 s. La matriz de reconstrucción de la imagen está constituida por 256 × 256 pixels (elementos gráficos), y el espesor de las «lonchas» que se analizan puede variar de 1,5 mm a 12 mm.*
Es claro que el tomógrafo computerizado puede utilizarse también como un aparato radiográfico normal para proyecciones normales y esta característica puede aprovecharse oportunamente para simplificar la gestión de todo el sistema. Efectivamente, el tomógrafo del ejemplo permite el posicionado automático del paciente bajo el scanner *utilizando una imagen radiográfica frontal producida de forma preliminar. La imagen frontal, presentada en el monitor, funciona como mapa de referencia para seleccionar el nivel exacto al cual se desea la imagen tomográfica. El Tomoscan puede realizar automáticamente en 6 segundos un escanograma frontal sobre un campo de 36 cm alrededor del punto interesado, detectanto durante la exploración la imagen radiográfica con tiras de 1,5 mm de longitud. Las tiras se recomponen en el monitor para formar la imagen frontal completa del campo y el operador puede seleccionar a continuación con el lápiz óptico la sección a evidenciar. El sistema de control computerizado del aparato procederá automáticamente al posicionado exacto del paciente. El Tomoscan también puede utilizarse para aprovechar al máximo su capacidad de detección desplazando radialmente el equipo móvil con respecto al centro de rotación, como se indica en el gráfico de al lado. Esta disposición permite adaptar la geometría del sistema a las dimensiones de la sección interesada, tanto si se trata del tórax de un adulto o del cráneo de un niño, aprovechando siempre completamente la matriz de detección. Cuando el tubo de radiación se acerca al paciente, la corriente de alimentación se reduce. El proceso se realiza automáticamente, controlado por el detector de referencia. El control automático de exposición permite efectuar exploraciones muy detalladas con dosis integradas de radiación muy reducidas.*
La amplia gama de las prestaciones que un tomógrafo puede ofrecer se debe en su mayor parte al software de gestión de todo el aparato. El sistema Tomoscan está controlado por un microprocesador de 16 bits PC857, dotado de memoria RAM de 128 kbytes. Al ordenador hay conectadas dos unidades de disco de 5,6 Mbytes cada una, que puede memorizar 30 imágenes y un disco flexible de 0,25 Mbytes, capaz de contener 10 imágenes. La memoria masiva está constituida por cinta magnética, que puede contener 300 imágenes de aproximadamente 730 m cada una.
El técnico en la consola principal está conectado permanentemente con el ordenador a través del teclado y del lápiz óptico. Con este último



Representación esquemática del generador de rayos X, del detector y del paciente en un aparato para la tomografía axial computerizada. Utilización óptima del detector obtenida mediante el desplazamiento radial del equipo móvil con respecto al campo de exploración.







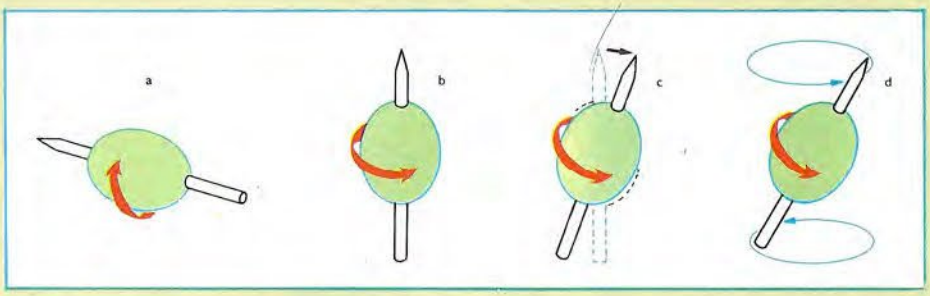
Philips

Tomografía axial computerizada. Arriba, a izquierda: vista de conjunto del Tomoscan y de las consolas de maniobra y proceso. Arriba: posicionado automático del paciente. Al lado: proceso del scanograma en consola remota.

puede seleccionar la imagen eligiendo entre una gran serie de posibilidades. Por ejemplo, puede obtenerse una ampliación 2X o 4X apuntando el lápiz a un punto cualquiera de la pantalla, que se interpretará como el centro de la zona a ampliar; o bien puede reclamarse simultáneamente en el monitor 4, 6 o 9 imágenes. Otra posibilidad es la de realizar un zoom en zonas circunscritas, o construir imágenes sobre planos de sección oblicuos, procesando las imágenes detectadas sobre una serie de planos axiales adyacentes. La definición de la imagen puede mejorarse actuando a voluntad sobre la escala tonal del monitor. Las imágenes obtenidas y procesadas se registran en la memoria masiva del sistema, y pueden reclamarse en línea incluso desde consolas diferentes a la principal, la cual tiene la misión de permitir la continuidad operativa de los costosos aparatos de exploración.

Además existe la posibilidad de difundir las imágenes a través de circuitos de televisión y de producir copias sobre papel en tiempo real. La inestimable utilidad de la tomografía axial computerizada no se detiene en la visualización de los detalles anatómicos. Las informaciones detectables también permiten planificar los tratamientos radioterápicos, gracias a las mediciones de densidad y de absorción de los tejidos, obtenibles directamente de las imágenes registradas por el tomógrafo. Así, la tecnología TAC permite hoy reducir al mínimo las dosis de radiación aplicadas a los pacientes sometidos a radioterapia, y que el ordenador gestione también los programas terapéuticos.

Una nueva metodología de investigación biomédica que permite revolucionar todavía más el sector del diagnóstico automatizado es la tomografía de resonancia magnética nuclear. *La resonancia magnética nuclear (NMR del inglés Nuclear Magnetic Resonance) es un fenómeno que interesa a los núcleos atómicos de algunos elementos, entre ellos el hidrógeno. Este último es particularmente importante a fines médicos, dado que las moléculas de agua (H_2O) están presentes en todos los tejidos vivientes.*



Representación esquemática del mecanismo de generación de la resonancia magnética nuclear.

El núcleo de un átomo de hidrógeno gira alrededor de su propio eje, y como está dotado de carga positiva, genera alrededor de él un campo magnético orientado ortogonalmente a la dirección de rotación (ver gráfico de arriba). Dada una cierta cantidad de hidrógeno, los núcleos que lo componen generan campos magnéticos orientados estadísticamente en todas las direcciones (a), pero si se genera un campo magnético externo, todos los núcleos tenderán a girar de manera que su eje de rotación se alinee con las líneas de fuerza del campo magnético aplicado (b). Como están girando, los núcleos se comportan como minúsculos giróscopos y, por tanto, tienen un movimiento de precesión alrededor de la dirección del campo magnético externo: cuando este último es homogéneo, todos los núcleos oscilan a la misma frecuencia. Si además del campo se aplica desde el exterior una excitación magnética pulsante de la misma frecuencia (estamos en el campo de las radiofrecuencias y, por tanto, será una señal radio), el eje de los núcleos en precesión tenderá a disponerse según un cierto ángulo con respecto al campo externo (c), y el movimiento de precesión quedará perturbado. Al cesar el impulso, los núcleos tienden a volver al estado inicial y emiten energía en forma de débiles señales radio a la frecuencia de precesión (d). La intensidad y la duración de la señal emitida dan una indicación sobre la densidad de los átomos sensibles a dicha frecuencia que hay en un cierto punto del espacio, y este comportamiento puede aprovecharse para el análisis de los tejidos.

El fenómeno de la resonancia magnética nuclear fue descubierto hace unos setenta años, pero sólo en 1977, en EE.UU., se intentó aprovecharlo para evidenciar la densidad de una vértebra. La síntesis de la imagen necesitó un tiempo de proceso de cuatro horas y media y el resultado fue una mancha oscura en un mar de niebla. Sin embargo, desde aquel momento, las principales industrias electrónicas conectadas con el sector biomédico se preocuparon de mejorar el método hasta hacer posible su aplicación extensiva. Los aparatos actualmente realizados tienen todavía una situación eminentemente experimental, pero se está procediendo rápidamente a las primeras realizaciones comerciales. Las principales ventajas de la metodología NMR que empujan a perseguir esta vía son la absoluta no invasividad de las técnicas de exploración y el hecho de que no se utilizan radiaciones ionizantes. La imagen se deriva de las señales radio emitidas por las sustancias que componen el organismo. La reconstrucción de las imágenes se efectúa con técnicas del todo análogas a las ya puestas a punto en el TAC, y esto permite la inmediata extensión de todo el software de aplicación desarrollado para este último. Además se ha aclarado suficientemente que las aplicaciones de los campos magnéticos necesarios para utilizar esta nueva técnica de exploración no tiene contraindicaciones y no presenta ningún efecto biológico. También se espera mucho de la posibilidad potencial que tiene la tomografía NMR de resolver y diferenciar los tejidos blandos.

Actualmente, Philips está perfeccionando una serie denominada Gyroscan, cuyos elementos fundamentales son:

— el imán principal, de tipo resistivo o superconductor;
— la batería de generadores de radiofrecuencia, que tienen funciones de excitación;
— la batería de detectores de radiofrecuencia, que tienen la función de detectar las señales de respuesta;
— el espectómetro universal, que sirve para el análisis de las señales de radiofrecuencia de respuesta;
— el ordenador y los periféricos.

Las notables ventajas enumeradas y la buena calidad de las imágenes producidas permiten esperar con fundado optimismo los futuros desarrollos de esta nueva técnica.

en agrupaciones más o menos grandes de instrucciones. Cada párrafo está identificado con un nombre perforado a partir del margen A (columna 8) y se cierra con un punto. La división en párrafos permite por un lado la posibilidad de reentrar en determinados puntos del programa mediante instrucciones de salto condicionado y, por otro, realizar todas las instrucciones en un cierto párrafo simplemente asociando el nombre al verbo PERFORM.
El nombre de cada párrafo lo crea el programador mediante una combinación de 30 (máximo) caracteres alfabéticos y números, más el signo menos con función de trazo de unión.
El Compilador considera terminado un párrafo cuando encuentra el nombre de otro párrafo.
A su vez, los párrafos pueden agruparse en una estructura de orden superior que es la sección (SECTION).
El nombre de las secciones se uniformizan con las mismas reglas válidas para los nombres de los párrafos, pero deben ir seguidos de la palabra reservada SECTION y del punto. Cada sección termina al final del programa, o donde se inicia otra.

ATENCION:

En la estructura del lenguaje Cobol es importante la disposición de las palabras reservadas en los diversos campos. Independientemente del periférico de entrada utilizado, cada programa puede imaginarse perforado en fichas. El correcto encolumnado de las palabras reservadas, o sea la correcta perforación de las instrucciones en las fichas, podrá deducirse del examen de los listados presentados de vez en cuando. Los ejemplos integrados en el texto, compuestos tipográficamente, sólo tienen la finalidad de presentar el orden de sucesión de las palabra-instrucción sencillas. Por exigencias tipográficas, la disposición de los diferentes campos puede que no respete la sintaxis de los listados.

A continuación se indican algunos ejemplos de nombres de párrafos y de secciones

```
INICIALIZA.                 (párrafo)
CALCULA-ITE SECTION.        (sección)
A 10-SUMA.                  (párrafo)
340-LEE-FILE.               (párrafo)
ESCRIBE SECTION.            (sección)
```

Obsérvese que los nombres, tanto de párrafos como de secciones, deben contener al menos un carácter alfabético.
La estructura general de la PROCEDURE DIVISION de un programa Cobol genérico organizada en secciones y párrafos es la siguiente:

```
PROCEDURE DIVISION.
PRIMERA SECTION.        (sección)
INICIALIZA.             (párrafo)
      ......    }
      ......    }       INSTRUCCIO-
      ......    }              NES
      ......    }
ABRE-FILES.             (párrafo)
      ......
      ......
      ......
      ..
SEGUNDA SECTION.        (sección)
LEE-FILE.               (párrafo)
      ......
      ......
      ......
      ......
PROCESA.                (párrafo)
      ......
      ......
      ......
TERCERA SECTION.        (sección)
IMPRIME.                (párrafo)
      ......
      ......
      ......
CIERRA-FILE.            (párrafo)
      ......
      ......
      ......
      ......
FIN.                    (párrafo)
STOP RUN.
```

Obsérvese que cada sección debe contener al menos un párrafo, y que el nombre de una sec-

ción debe ir seguida del nombre de un párrafo; es decir, es erróneo escribir por ejemplo,

```
PRIMERA SECTION.        (sección)
MOVE 0 TO A.            (instrucción)
```

puesto que debe escribirse

```
PRIMERA SECTION.        (sección)
INICIALIZA.             (párrafo)
MOVE 0 TO A.            (instrucción)
```

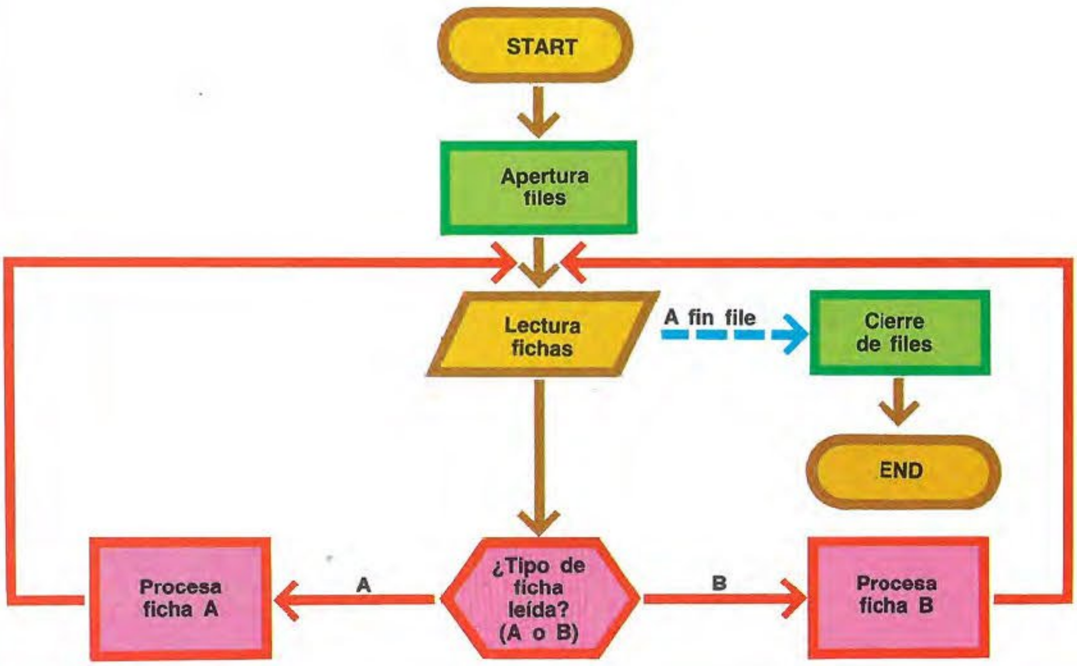
Del mismo modo, cada párrafo debe contener al menos una instrucción. Como es posible observar en los ejemplos indicados hasta ahora, los nombres de los párrafos o de las secciones tienen todos un evidente significado autoexplicativo. La asignación de los nombres de forma significativa, o bien de manera que el propio programador o un eventual lector pueda intuir la función de proceso, no constituye una regla del Cobol, pero contribuye a facilitar la lectura y la manutención de cada programa.

Ejemplo de estructuración de una PROCEDURE DIVISION

Supóngase que hay que escribir un programa que lea de un file dos tipos de fichas diferentes,

DIAGRAMA DE FLUJO





ESTRUCTURACION DE UNA PROCEDURE DIVISION. DESCRIPCION DE LOS FILES FICHAS DE ENTRADA

```
01      FICHA-B.
        05  FILLER                       PIC X.
        05  NOMBRE-APELLIDO              PIC X(30).
        05  SALDO                        PIC 9(10).
        05  INTERES                      PIC (3)V9(3).
        05  GASTOS                       PIC 9(5).
        05  FILLER                       PIC X(5).

01      FICHA-A.
        05  FILLER                       PIC X.
        05  NOMBRE                       PIC X(30).
        05  FECHA-NACIMIENTO
            10    DIA                       PIC 9(2).
            10    MES                       PIC 9(2).
            10    AÑO                       PIC 9(2).
        05  DEBE                            PIC 9(10).
        05  HABER                           PIC 9(10).
```

DESCRIPCION DE LA LINEA DE IMPRESION

```
01      LINEA.
        05  FILLER                          PIC X(12) VALOR 'NOMBRE'.
        05  NOMBRE-LINEA                    PIC X(30).
        05  FILLER                          PIC X(10) VALUE SPACES.
        05  FILLER                          PIC X(08) VALUE "NACIDO EL'.
        05  FECHA LINEA                     PIC X(2)/X(2)/X(2).
        05  FILLER                          PIC X(10) VALUE '    DEBE'.
        05  DEBE-LINEA                      PIC **********.
        05  FILLER                          PIC X(10) VALUE '    HABER'.
        05  HABER-LINEA                     PIC **********.
        05  FILLER                          PIC X(10) VALUE 'SALDO'.
        05  SALDO-LINEA                     PIC **********.
```

DESCRIPCION DEL RECORD DE SALIDA

```
01      RECORD-B.
        05  NOMBRE-RECORD                   PIC X(30).
        05  TOTAL-RECORD                    PIC 9(10).
```

A y B, con las descripciones indicadas en el listado de arriba.
Además supongamos que por cada tarjeta de tipo A leída debe crearse e imprimirse la línea descrita en el listado del centro, mientras que, para cada tarjeta de tipo B, debe prepararse y escribirse en el file el record descrito en el listado de arriba. El diagrama de flujo del programa se ha indicado en la página anterior, junto con la diagramación del flujo de los datos.
El proceso de las fichas de tipo A prevé las siguientes operaciones:

1 / Calcular en un campo de trabajo, DISPONIBLE-SALDO, la diferencia entre DEBE y HABER
2 / Situar el valor así obtenido en el campo SALDO-LINEA
3 / Situar en los correspondientes campos de LINEA todos los otros campos de la FICHA-A
4 / Imprimir la línea así compuesta.

En cambio, para el proceso de las fichas de tipo

B, las operaciones serán las siguientes:

1 / Calcular el resultado de la fórmula en el campo de trabajo DISPONIBLE-TOTAL

SALDO + INTERES – GASTOS

2 / Colocar el valor obtenido en el campo TOTAL-RECORD
3 / Colocar el campo NOMBRE-APELLIDO en NOMBRE-RECORD
4 / Escribir el record en el file FILE-OUT.

Es necesario indicar que en el desarrollo de este ejemplo aparecen instrucciones y verbos todavía no tratados, aunque su interpretación es inmediata; sin embargo, este ejemplo sólo tiene la función de indicar las ventajas asociadas al uso de los párrafos y de las secciones en el planteo de un programa. La PROCEDURE DIVISION del programa examinado podría estructurarse como en el listado de abajo.
No obstante, escribir un programa de esta manera comporta diversos inconvenientes:

- El Compilador podría no conseguir analizar correctamente frases tan largas como las escritas para cada rama de la instrucción IF.
- Para programas de complejidad superior a la del descrito, el recurso de un número elevado de saltos condicionados o incondicionados (GO TO) es posible que no haga evidente de forma inmediata la determinación de ciclos infinitos indeseados.
- Se reduce drásticamente la posibilidad de intuir rápidamente las funciones de una determinada rama del programa y, en consecuencia, del propio programa.

A fin de evitar los inconvenientes citados, la PROCEDURE DIVISION de dicho programa puede detallarse recurriendo a las secciones, como se muestra en el listado de las págs. 991 y 992.
Como puede observarse, la PROCEDURE DIVISION queda ahora reducida a las pocas líneas de la PRIMERA SECTION, de cuya lectura es fácil intuir las funciones y los enlaces de las diversas ramas que sigue al programa. Todas las

ESTRUCTURACION DE UNA PROCEDURE DIVISION (EJEMPLO 1)

```
PROCEDURE DIVISION.
INICIO.
     OPEN INPUT FICHAS.
     OPEN OUTPUT IMPRESION.
     OPEN OUTPUT FILE-OUT.
LEE-FICHA.
     READ FICHAS
          AT END
          CLOSE FICHAS
          CLOSE IMPRESION
          CLOSE FILE-OUT
          GO TO FIN.
     IF   TIPO-FICHA IS EQUAL TO 'A'
          MOVE FICHA TO FICHA-A
          COMPUTE DISPONIBLE-SALDO = HABER - DEBE
          MOVE DISPONIBLE-SALDO TO SALDO-LINEA
          MOVE NOMBRE            TO NOMBRE-LINEA
          MOVE FECHA-NACIMIENTO TO FECHA-LINEA
          MOVE DEBE              TO DEBE-LINEA
          MOVE HABER             TO HABER-LINEA
          WRITE LINEA
          GO TO LEE-FICHA.
     IF   TIPO-FICHA IS EQUAL TO 'B'
          MOVE FICHA TO FICHA-B
          COMPUTE DISPONIBLE-TOTAL = SALDO + INTERES - GASTOS
          MOVE DISPONIBLE TOTAL TO TOTAL-RECORD
          MOVE NOMBRE-APELLIDO TO NOMBRE-RECORD
          WRITE RECORD-B
          GO TO LEE-FICHA.
FIN.
     STOP RUN.
```

ESTRUCTURACION DE UNA PROCEDURE DIVISION (EJEMPLO 2)

```
PROCEDURE DIVISION
PRIMERA SECTION.
INICIO.
   PERFORM ABRE-FILES.
   PERFORM PRIMERA-LECTURA.
PROCESA.
   IF  TIPO-FICHA IS EQUAL TO 'A'
       PERFORM PROCESA-A
       WRITE LINEA.
   IF  TIPO-FICHA IS EQUAL TO 'B'
       PERFORM PROCESA-B
       WRITE RECORD-B.
   PERFORM LEE-FICHA.
   GO TO PROCESA.
*
*
*
*
*
ABRE-FILES SECTION.
ABRE.
   OPEN INPUT FICHAS.
   OPEN OUTPUT IMPRESION.
   OPEN OUTPUT FILE-OUT.
FIN-ABRE. EXIT.
*
*
*
*
*
CIERRA-FILES SECTION.
CIERRA.
   CLOSE FICHAS.
   CLOSE IMPRESION.
   CLOSE FILE-OUT.
FIN-CIERRA. EXIT.
*
*
*
*
*
PRIMERA LECTURA SECTION.
PRIMERA-FICHA.
     READ FICHAS
          AT END
          DISPLAY '** FILE FICHAS VACIO **'
                  UPON PRINTER
                  PERFORM FIN-PROCESA.
FIN-PRIMERA-FICHA. EXIT.
*
*
*
*
*
PROCESA-A SECTION.
PROC-FICHA-A.
          MOVE FICHA TO FICHA-A.
          COMPUTE DISPONIBLE-SALDO = HABER - DEBE.
          MOVE DISPONIBLE - SALDO TO SALDO-LINEA.
          MOVE NOMBRE             TO NOMBRE-LINEA.
          MOVE FECHA NACIMIENTO   TO FECHA-LINEA.
          MOVE DEBE               TO DEBE-LINEA.
          MOVE HABER              TO HABER-LINEA.
FIN-PROC-FICHA-A. EXIT.
*
*
*
```

```
*
*
 PROCESA-B SECTION.
 PROC-FICHA-B.
             MOVE FICHA TO FICHA-B.
             COMPUTE DISPONIBLE-TOTAL = SALDO + INTERES - GASTOS.
             MOVE DISPONIBLE-TOTAL TO TOTAL-RECORD.
             MOVE NOMBRE-APELLIDO TO NOMBRE-RECORD.
 FIN-PROC-FICHA-B. EXIT.
*
*
*
*
*
 LEE-FICHA SECTION.
 OTRAS-FICHAS.
    READ FICHAS
          AT END
          PERFORM FIN-PROC.
 FIN-OTRAS-FICHAS. EXIT.
*
*
*
*
*
 FIN-PROC SECTION.
 FIN.
     PERFORM CIERRA-FILES.
     DISPLAY '*** FIN PROCESO ***'
          UPON PRINTER.
     STOP RUN.
*
*
*
*
*
```

demás secciones, escritas después de la PRIMERA SECTION, en realidad se reclaman y realizan cada vez mediante el verbo PERFORM.
Se volverá ampliamente sobre el uso de la instrucción PERFORM y sobre las modalidades de su utilización.

Las instrucciones del Cobol

Antes de analizar en detalle el lenguaje Cobol deben conocerse algunas convenciones y restricciones impuestas al programador.
El conjunto de caracteres admitidos por el Compilador está constituido por todas las letras del alfabeto inglés desde la A a la Z, por todos los números naturales de 0 a 9 y por un conjunto restringido de caracteres especiales, a los cuales el Compilador asocia los significados indicados en la tabla de la derecha.
El Cobol, por ser un lenguaje de tipo descriptivo, utiliza palabras enteras y abreviadas de la lengua inglesa, a las que están asociadas determinadas acciones del Compilador. Estas palabras (cerca de 250) no las puede utilizar el programador de forma diferente a la prevista por la sintaxis del lenguaje, ni puede alterarse su ortografía con la omisión o la inserción de caracteres. Por esto se llaman **palabras reservadas.** Por ejemplo, la palabra reservada SOURCE-COMPUTER no puede escribirse SOURCE COMPUTER o SOURCECOMPUTER, puesto que el

Compilador no la reconocería, ni el programador puede escribir un programa en que la palabra SOURCE-COMPUTER se utilice como nombre de una variable o de un file. El siguiente ejemplo muestra el uso erróneo de la palabra reservada DATA, utilizada como nombre de un campo definido por el programador:

```
01 DATA.
   05 DIA          PIC 99.
   05 MES          PIC 99.
   05 AÑO          PIC 99.
```

A continuación se indica otro ejemplo en el que se presentan algunas palabras reservadas

```
PROCEDURE DIVISION.
INICIO SECTION.
INICIA.
  OPEN INPUT FILE-IN.
  OPEN INPUT FICHAS.
  OPEN OUTPUT FILE-OUT.
  PERFORM CONTROLA-FICHAS.
  IF ERROR = 1
  GO TO FIN.
PROCESA.
  READ FILE-IN INTO REC-IN
            AT END
FIN.
   CLOSE FILE-IN
         FILE-OUT
```

Para hacer después discursivo el lenguaje, en algunas secuencias de caracteres de uso más frecuente se han asociado palabras reservadas particulares: las **constantes figurativas.** Sus significados se indican en la tabla que sigue, donde se han encerrado en la misma casilla los formatos equivalentes de dicha constante.

Constante figurativa	**Caracteres equivalentes**
ZERO ZEROS ZEROES	Uno o más ceros numéricos, o también uno o más caracteres 0 según el contexto.
SPACE SPACES	Uno o más espacios (blank).
LOW-VALUE LOW-VALUES	Uno o más caracteres con el peso menor de la secuencia del código utilizado.
HIGH-VALUE HIGH-VALUES	Uno o más caracteres con el peso mayor de la secuencia del código utilizado.
QUOTE QUOTES	Una o más comillas (").
ALL 'carácter-deseado'.	Uno o más caracteres 'carácter-deseado' hasta el llenado del campo receptor.

Considérese un campo genérico (de nombre CAMPO) que pueda albergar hasta seis caracteres.

Símbolo	**Descripción**	**Significado para el Compilador**	**Categoría del símbolo**
.	Punto	Fin de un período	Punteado
,	Coma	Separador de una serie de operandos	Punteado
;	Punto y coma	Separador de una serie de cláusulas	Punteado
'	Acento	Delimitador de una cadena de caracteres	Punteado
( )	Paréntesis	Delimitador de índices	Punteado
+	Más	Suma	Aritmético
−	Menos	Resta	Aritmético
*	Asterisco	Multiplicación	Aritmético
/	Barra (slash)	División	Aritmético
**	Dos asteriscos	Elevación a potencia	Aritmético
=	Igual	Igual	Aritmético
>	Mayor	Mayor	De condición
<	Menor	Menor	De condición
( )	Paréntesis	Delimitador de expresiones. También controla la secuencia de ejecución de operaciones lógicas y aritméticas.	De condición

En la tabla de abajo se ha ilustrado el efecto de la transferencia en CAMPO de las constantes figurativas relacionadas.

Análogamente a lo que sucede en los lenguajes humanos, cada acción que el calculador debe realizar se pide mediante **verbos.** Cada verbo que encuentra el Compilador en el programa fuente se traduce en una secuencia de instrucciones en lenguaje máquina aptas para actuar sobre los componentes hardware llamados a causa de dicha instrucción. Se intuye que todos los verbos del Cobol son palabras reservadas, que el programador no puede utilizar si no es en la forma sintáctica y para las finalidades específicamente previstas.

Los verbos del Cobol pueden subdividirse en cinco categorías, según el tipo de función que pueden realizar.

Verbos de I/O. La notación I/O es la abreviatura de Input/Output. A esta categoría pertenecen los verbos que permiten tansferir datos de y hacia diferentes unidades periféricas del calculador:

OPEN
CLOSE
READ
WRITE
ACCEPT
DISPLAY

Verbos aritméticos. Permiten actuar sobre constantes y campos numéricos con las operaciones fundamentales de la artimética:

COMPUTE
ADD
SUBTRACT
MULTIPLY
DIVIDE

Verbos de transferencia o de manipulación de los datos. Permiten el desplazamiento del contenido de un campo, elemental o compuesto, de una posición de memoria a otra (transferencia), la composición de más campos en un campo único y viceversa, así como operaciones particulares sobre un determinado dato:

MOVE
INSPECT
STRING
UNSTRING

Verbos para el control de la secuencia de las instrucciones. Son verbos que imprimen a la ejecución del programa un proceso diferente al del estrictamente secuencial que refleja la disposición de las instrucciones en el programa fuente:

GO TO
PERFORM
EXIT
STOP

Verbos condicionales. Permiten analizar la verificación o no de un determinado suceso:

IF
SEARCH

Verbos de I/O

Estos verbos preceden a la transferencia de datos entre la memoria central del procesador y las unidades externas, como lectores de fichas, impresoras, unidades de cinta y disco. Para poder trabajar sobre estas unidades mediante verbos o preposiciones de input/output, es necesario que el programador haya declarado al Com-

TRANSFERENCIA DE CONSTANTES FIGURATIVAS

MOVE ZEROS	TO CAMPO.	0	0	0	0	0	0
MOVE SPACES	TO CAMPO.						
MOVE LOW-VALUES	TO CAMPO.	A	A	A	A	A	A
MOVE HIGH-VALUES	TO CAMPO.	Z	Z	Z	Z	Z	Z
MOVE QUOTES	TO CAMPO.	"	"	"	"	"	"
MOVE ALL '*'	TO CAMPO.	*	*	*	*	*	*

pilador que en el curso del proceso accederá a los periféricos. Ya se ha indicado que a esta función le precede la INPUT-OUTPUT SECTION de la ENVIRONMENT DIVISION, en la que se han declarado tanto los nombres de los files como los dispositivos en que residen. Sin embargo, en esta sección, no se proporciona ninguna información acerca de la modalidad con que el programa accederá a los files. Es decir, lo escribe en la ENVIRONMENT DIVISION.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT FILE-A ASSIGN TO DISC FILE-A
```

no declara si el FILE-A se utilizará en el curso del programa en lectura o en escritura. Recordemos que la ENVIRONMENT DIVISION, al ir delante de la definición de las interaciones con el procesador usado, puede tener formatos diferentes de calculador a calculador.*

Las únicas excepciones a estas reglas son los files asignados al lector de fichas (CARD-READER), a la impresora (PRINTER) y al perforador de fichas (CARD-PUNCH), para los que es implícita la modalidad de uso. Efectivamente, es obvio que un lector de fichas puede utilizarse exclusivamente como unidad de entrada, y una impresora, o un perforador de fichas, sólo como unidad de salida.

Para hacer clara la serie de operaciones que debe efectuarse sobre un file Cobol para utilizar su contenido, es posible hacer referencia al siguiente ejemplo.

Supóngase que se quiere actualizar la propia agenda (FILE = AGENDA) que está en la cassette de las anotaciones (UNIDAD = ANOTACIONES). La primera operación a hacer será naturalmente la de abrir la agenda; la siguiente operación será la de consultar (OUTPUT) las notas del día. En el momento en que se abre la agenda ya se tendrá en mente la realización de esta operación (ABRIR PARA ESCRIBIR = OPEN OUTPUT).

Terminada la actualización se cerrará la agenda (CERRAR AGENDA = CLOSE AGENDA).

En el intento de mantener el paralelo con el ejemplo indicado, supóngase que el procesador está dotado de una unidad llamada ANOTACIONES; en este caso, la secuencia de operaciones escritas podría traducirse en el listado de la página siguiente.

Apertura y cierre de los files: los verbos OPEN y CLOSE

Antes de continuar es interesante ilustrar cuál es el mecanismo activado por las operaciones de apertura y cierre de uno o más files en el interior de un programa.

En el momento en que un programa en fase de ejecución encuentra la instrucción OPEN, deja disponible en memoria el área (buffer) reservada a las operaciones en que el file controla o crea las «labels» de cabecera y posiciona la unidad física en la que reside el file al inicio del mismo.

Debe subrayarse que la operación de apertura de un file posiciona la unidad en el primer record del file, pero no deja disponible su contenido. Para poder transferir de o hacia el file el contenido del record es necesario leerlo o escribirlo, o bien especificar explícitamente la acción mediante el verbo READ o WRITE.

Generalmente, a cada programa se le asigna una región de memoria cuya extensión no comprende las áreas reservadas a las acciones de lectura y escritura en files declarados en la ENVIRONMENT DIVISION y descritos en la DATA DIVISION.

Estas áreas se alojan en memoria en el acto de la apertura del file.

Por este motivo puede suceder que durante la ejecución de un programa, éste termine en error por violación de la capacidad de memoria sólo durante la operación de apertura de un file. Como un file puede abrirse o cerrarse un número cualquiera de veces durante el mismo programa, es preferible, si no intervienen exigencias de proceso particulares, cerrar el file inmediatamente después de su utilización.

Como ya se ha dicho, la apertura de un file presupone también la declaración de la modalidad en que se utilizará dicho file. Es decir, el Cobol permite abrir un file haciéndolo accesible a operaciones de:

Lectura	(INPUT)
Escritura	(OUTPUT)
Lectura-escritura	(I-O)
Puesta en fila	(EXTEND)

* En este caso se ha utilizado el formato típico de los sistemas UNIVAC de la serie 1100 (Sperry Corporation), así como para una inmediata interpretación respecto a los otros sistemas.

EJEMPLO DE APERTURA DE UN FILE

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. AGG-AGENDA.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. ...........
 OBJETC-COMPUTER. ...........
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT AGENDA ASSIGN TO ANOTACIONES.
 DATA DIVISION.
 FILE SECTION.
 FD  AGENDA
        LABEL RECORD STANDARD.
 01     NOTA.
        05  .............
            .............
            .............
            .............
            .............
*
*
*
*
*
 PROCEDURE DIVISION.
 PRIMERA SECTION.
 ABRE-FILE.
        OPEN OUTPUT AGENDA.
        PERFORM ESCRIBE-NOTA.
        CLOSE AGENDA.
        STOP RUN.
*
*
*
*
*
 ESCRIBE-NOTA SECTION.
 ESCRIBE.
           ..............
           ..............
           ..............
           ..............
        ESCRIBE NOTA.
 FIN-ESCRITURA. EXIT.
*
*
```

Utilización en lectura: OPEN INPUT. El formato de la instrucción OPEN en esta modalidad se indica en la tabla de la página siguiente.
Las cláusulas opcionales REVERSED y WITH NO REWIND se refieren a files residentes en cinta. Escribiendo simplemente

OPEN INPUT FILE-A.

cualquiera que sea la unidad a que está asignado el file FILE-A se tiene la apertura del file y el posicionado de la unidad en el primer record. En el caso de un file asignado a la unidad de cinta, la cinta se rebobina hasta su punto de partida (LOAD POINT), a menos que se utilice una de las dos cláusulas. Usando la cláusula REVERSED se tiene el posicionado de la unidad en el último record del file, de manera que pueda leerse en sentido directo, mientras la cláusula WITH NO REWIND impide el rebobinado de la cinta. A continuación se indicará un ejemplo correspondiente al uso de las dos cláusulas.

Utilización en escritura: OPEN OUTPUT. La apertura de un file en escritura se obtiene con el formato de la instrucción OPEN indicado al pie de esta página. Las funciones de la cláusula WITH NO REWIND es la misma descrita para la lectura. Obsérvese que abrir un file en OUTPUT significa predisponer el periférico a escribir partiendo del primer record del file: el eventual contenido del record será cubierto por las siguientes operaciones de escritura (WRITE).
A título de ejemplo, supóngase que debe leerse el file F1-CINTA en cinta para copiar su contenido, record por record, sobre el file F-DISCO residente en disco. Este último file deberá copiarse seguidamente en cinta como F2-CINTA a continuación de F1-CINTA. Supongamos además que el file F2-CINTA deba tener como primer record el último record de F1-CINTA. El programa podrá organizarse y detallarse como se muestra en la pág. 998 y en el listado de la pág. 999.
Las secciones contenidas en el programa no se han declarado, puesto que implican el uso de los verbos READ y WRITE (que todavía no se han descrito), pero su nombre es indicativo de la función que realizan.
Habiendo abierto el file F1-CINTA en la modalidad INPUT con la cláusula REVERSED, la primera operación de lectura se realizará en el último record del file. Este record se lee y se salva en un área de trabajo (DISPONIBLE-REC-F1) de la que después se transferirá al file F2-CINTA como primer record.
Como debe copiarse todo el F1-CINTA en F-DISCO ahora es necesario disponer del file F1-CINTA desde su primer record; es decir, debe rebobinarse la cinta hasta el LOAD POINT.
Para obtener esto es suficiente cerrar el file y reabrirlo sin añadir otras cláusulas (CLOSE, F1-CINTA, OPEN INPUT F1-CINTA). En cambio, la instrucción OPEN OUTPUT F-DISCO abre el file F-DISCO y lo habilita para reservar datos en escritura.
La operación de lectura de F1-CINTA y de escritura en F-DISCO la efectúa, record por record, la SECTION LEE-F1-ESCRIBE-F.
Después de haber leído F1-CINTA hasta el final, la unidad se posiciona sobre el carácter de fin-file de F1-CINTA. Como el programa ahora debe copiar el contenido de F-DISCO en F2-CINTA en posición consecutiva a F1-CINTA, debe cerrar F1-CINTA sin rebobinar la cinta y, por tanto, abrir F2-CINTA. Esto se obtiene añadiendo la cláusula WITH NO REWIND a las dos instrucciones citadas.
La última parte del programa es de interpretación inmediata, puesto que las operaciones sucesivas son:

- copia el campo DISPONIBLE-REC-F1, en el que se salva el contenido del último record de F1-CINTA como primer record de F2-CINTA.
- copia el contenido de F-DISCO en F2-CINTA (PERFORM LEE-F-ESCRIBE-F2) record por record hasta el final del file
- cierra todos los files todavía abiertos y rebobina la cinta (CLOSE F-DISCO, F2-CINTA).

Utilización en lectura-escritura: OPEN I-O
El file direccionado debe residir en disco y en él pueden efectuarse tantas operaciones de lectura como de grabación. El formato es:

OPEN I-O nombre-de-file.

Puesta en fila: OPEN EXTEND. Mediante esta instrucción el file, que puede residir tanto en disco como en cinta, en el acto de la apertura se

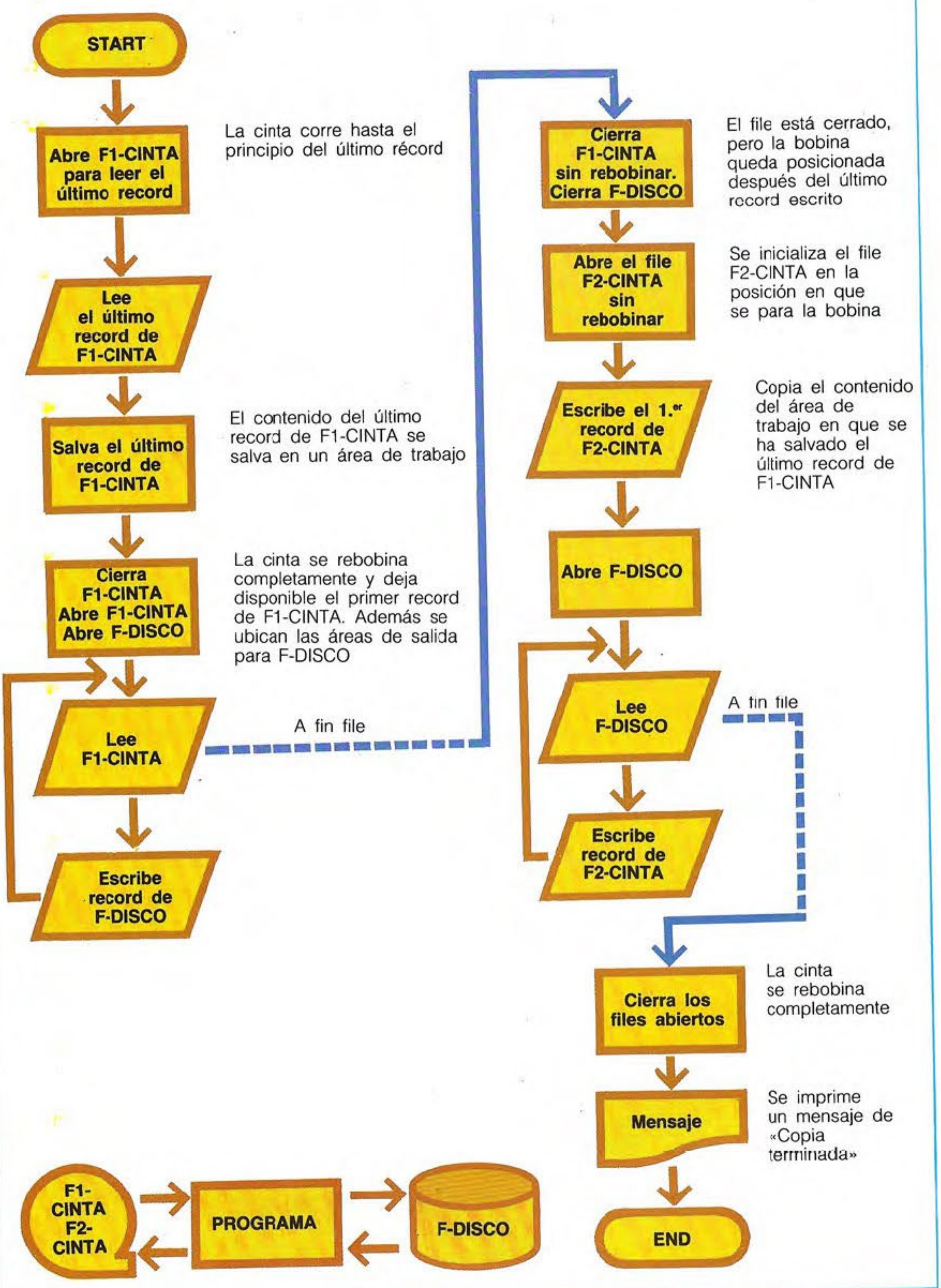
FORMATO DE LA INSTRUCCION OPEN EN LECTURA

OPEN INPUT nombre-de-file { REVERSED | WITH NO REWIND }

FORMATO DE LA INSTRUCCION OPEN EN ESCRITURA

OPEN OUTPUT nombre-de-file [WITH NO REWIND]

EJEMPLO DE USO DE LOS VERBOS OPEN Y CLOSE



EJEMPLO DE USO DE LOS VERBOS OPEN Y CLOSE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COPIA.
REMARKS. ESTE PROGRAMA ES UN EJEMPLO DE USO
         DE LAS INSTRUCCIONES OPEN Y CLOSE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT DIVISION.
FILE-CONTROL.
     SELECT F1-CINTA   ASSIGN  TO TAPE F1.
     SELECT F2-CINTA   ASSIGN  TO TAPE F2.
     SELECT F-DISCO    ASSIGN  TO DISC F-DISCO.
DATA DIVISION.
FILE SECTION.
FD  F1-CINTA
    LABEL RECORD STANDARD.
01  REC-F1-CINTA.
    05  ..............
    05  ..............
FD  F2-CINTA
    LABEL RECORD STANDARD.
01  REC-F2-CINTA.
    05  ..............
    05  ..............
FD  F-DISCO
    LABEL RECORD STANDARD.
01  REC-F-DISCO.
    05  ..............
    05  ..............
*
WORKING-STORAGE SECTION.
01  DISPONIBLE-REC-F1              PIC X(...).
*
PROCEDURE DIVISION.
PRIMERA SECTION.
INICIO.
    DISPLAY '** INICIO COPIA **' UPON PRINTER.
    OPEN INPUT F1-CINTA REVERSED.
    PERFORM LEE-ULTIMO-REC-F1.
    CLOSE F1-CINTA.
    OPEN INPUT F1-CINTA.
    OPEN OUTPUT F-DISCO.
    PERFORM LEE-F1-ESCRIBE-F.
    CLOSE F1 CINTA WITH NO REWIND.
    CLOSE F-DISCO.
    OPEN  F2-DISCO WITH NO REWIND.
    OPEN INPUT F-DISCO.
    PERFORM COPIA-ULTIMO-REC-F1.
    PERFORM LEE-F-ESCRIBE-F2.
    CLOSE F-DISCO.
    CLOSE F2-CINTA.
    DISPLAY '** COPIA TERMINADA **'
            UPON PRINTER.
    STOP RUN.
*
* = = = = = =   SECCIONES   = = = = = = = = =
*
LEE-ULTIMO-REC-F1 SECTION.
ULTIMO-REC.
     ..............
ULTIMO-REC-EX. EXIT.
*
LEE-F1-ESCRIBE-F SECTION.
F1-F.
     ..............
F1-F-EX. EXIT.
*
COPIA-ULTIMO-REC-F1 SECTION.
COPIA-ULTIMO.
     ..............
COPIA-ULTIMO-EX. EXIT.
*
LEE-F1-ESCRIBE-F2 SECTION.
F1-F2.
     ..............
F1-F2-EX. EXIT.
```

desplaza hasta su último record lógico y se predispone para la escritura de otros records de esta posición en adelante. El formato es:

OPEN EXTEND nombre-de-file

Hay que observar la gran utilidad de esta instrucción para realizar introducciones de datos al final de files ya existentes.
En tal caso, operar del siguiente modo:

```
OPEN I-O FILE-A.
PERFORM LEE-TODO-FILE-A.
PERFORM ESCRIBE-NUEVO-RECORD.
CLOSE FILE-A.
```

Utilizando la cláusula EXTEND, las mismas operaciones se realizarían con la secuencia

```
OPEN EXTEND FILE-A.
PERFORM ESCRIBE-NUEVO-RECORD.
CLOSE FILE-A.
```

Es evidente que la ventaja es tanto más sensible cuanto mayor es el número de records contenidos en el file.
La cláusula EXTEND no puede utilizarse para un file vacío: el file abierto en modalidad EXTEND ya debe contener al menos un dato.

En la tabla de abajo se han indicado los formatos completos de las instrucciones OPEN y CLOSE. Obsérvese que es posible cerrar un file para impedir su reapertura en el curso del programa utilizando la cláusula LOCK con la CLOSE.
A fin de conseguir una mayor legibilidad del programa además es aconsejable escribir las instrucciones columnándolas como sigue

```
OPEN INPUT      FILE-1
                FILE-2
                FILE-3
     OUTPUT     FILE-4
                FILE-5
..........................................
CLOSE  FILE-1
       FILE-2 WITH LOCK
       ..........
       FILE-5
```

Lectura y escritura en files: los verbos READ y WRITE

En muchos de los ejemplos ilustrados hasta ahora se ha indicado que una operación de lectura en file se efectúa con la instrucción READ, mientras que la escritura de un record en un file se obtiene mediante el verbo WRITE.
El formato más sencillo de las dos instrucciones es el siguiente

READ nombre-de-file AT END frase-imperativa.
WRITE nombre de record.

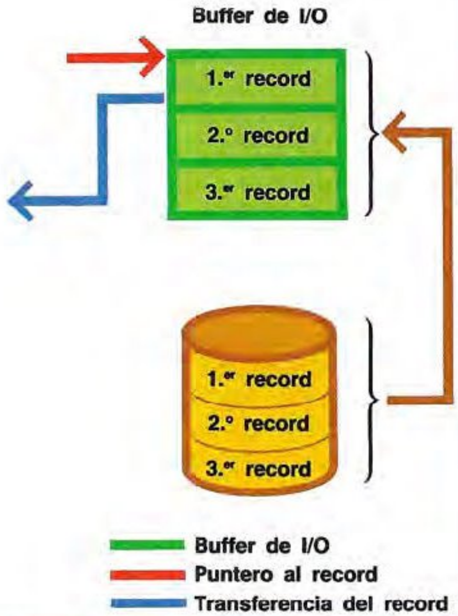
Es interesante subrayar que se lee (READ) un file y se escribe (WRITE) un record.
Cada instrucción de lectura en un file deja disponible para el proceso un record lógico. Para los files secuenciales pueden efectuarse, después de la apertura y del cierre del file, tantas lecturas como records hayan presentes en el file. A continuación, a cada comando de lectura se efectúa automáticamente un test para verificar si el file se ha acabado o no. Entonces, el sistema operativo reconoce el final del file y a

FORMATOS COMPLETOS DE LAS INSTRUCCIONES OPEN Y CLOSE

OPEN	INPUT	{nombre-de-file}	[{REVERSED / WITH NO REWIND}]	....
	OUTPUT	nombre-de-file	[{WITH NO REWIND}]	....
	I-O	{nombre-de-file}	....	
	EXTEND	nombre-de-file	....	
CLOSE	nombre-de-file		[WITH {NO REWIND / LOCK}]	....

MECANISMO DE LECTURA DE UN FILE

```
FD FILE-A
    LABEL RECORD STANDARD
    BLOCK CONTAINS 3 RECORDS
    RECORD CONTAINS 100 CHARACTERS
    01 REC-A                    PIC x(100)
    WORKING-STORAGE SECTION
    01 REC-A-W-S                PIC x(100)
    PROCEDURE DIVISION
    MAIN SECTION
    ABRE-FILE
        OPEN INPUT FILE-A
        READ FILE-A INTO REC-A-W-S
            AT END
            DISPLAY 'FIN FILE-A'
                    UPON PRINTER
            CLOSE FILE-A
```



continuación la frase-imperativa descrita en la cláusula AT END. La instrucción READ puede utilizarse también en el siguiente formato:

<u>READ</u> nombre-de-file <u>INTO</u> nombre-de-dato
<u>AT END</u> frase-imperativa

Mientras en el primer formato el contenido del record se deja disponible al programa mediante el área descrita en FILE SECTION, en el segundo formato el mismo record se copia en un área definida en la WORKING-STORAGE SECTION.
Para ilustrar el mecanismo de lectura de un file considérese el sencillo caso, representado arriba, del file FILE-A constituido por seis records de 100 caracteres de longitud fija y con un factor de bloqueo igual a 3.
Antes de la apertura del file, el contenido del área REC-A es completamente imprevisible.
La apertura del file comporta la ubicación en memoria de un área (buffer de entrada) dimensionada para contener 3 records de 100 caracteres cada uno, o sea el resultado de la lectura del primer bloque de tres records en el buffer de entrada. En este momento, el contenido de REC-A todavía es indefinido. La primera lectura (READ) no efectúa por sí misma ningún desplazamiento de datos de memoria del buffer de entrada hacia REC-A, sino que se limita a crear un puntero entre REC-A y el primer record en el buffer, dejando de este modo disponible al programa el contenido del record. Además, como se ha especificado la cláusula INTO, el contenido del primer RECORD se copia en REC-A-W-S.
La segunda lectura hace apuntar REC-A al segundo record y copia su contenido del buffer de entrada en REC-A-W-S.
La tercera lectura se comporta análogamente.
La cuarta lectura, al encontrar el buffer de entrada vacío, procede a leer del file el segundo bloque y hace accesible al programa el primer record del buffer, efectuando como siempre la copia en REC-A-W-S. Una lectura sucesiva a la sexta encuentra la «mark» de fin-file y, por tanto, realiza la frase imperativa.

```
DISPLAY 'FIN FILE-A'
        UPON PRINTER
CLOSE   FILE-A
```

El cierre del file comporta la liberación del buffer de entrada y, en consecuencia, el contenido de REC-A ya no está disponible. El proceso descrito es perfectamente análogo para la instrucción WRITE, pero se recorre en sentido inverso.

En el formato

WRITE nombre-de-record

la instrucción efectúa las operaciones de escritura apuntando directamente al buffer de salida, mientras que con la forma

WRITE nombre-de-record FROM nombre-de-dato

en el buffer de salida se copia el contenido del campo de WORKING-STORAGE nombre-de-dato. Debe observarse explícitamente que la cláusula INTO para la instrucción READ y la cláusula FROM para la WRITE son, a todos los efectos, desplazamientos de datos de un punto a otro de la memoria y, por tanto, respetan las reglas que rigen este tipo de operación.
Sin afrontar un análisis detallado de estas reglas, que se tratarán a fondo al hablar de la instrucción MOVE, es interesante anticipar algunas nociones útiles para la comprensión de las instrucciones que se examinan. Si en una instrucción READ el nombre-de-dato que acompaña la cláusula INTO tiene una longitud inferior a la descripción del record a leer, el record resultará truncado a la derecha. Considérese el ejemplo:

```
DATA DIVISION.

FILE SECTION.
FD FICHAS
LABEL RECORD OMITTED.
01 FICHA                        PIC X(80).

WORKING-STORAGE SECTION.

01 FICHA-W-S.
   05 TIPO-FICHA                PIC X(2).
   05 FECHA-FICHA.
      10 DIA                    PIC 9(2).
      10 MES                    PIC 9(2).
      10 AÑO                    PIC 9(2).
   05 ARTICULO.
      10 SERIE                  PIC X(2).
   10 NUMERO                    PIC 9(4).
   05 FILLER                    PIC X(3).
```

(puede verse que el record FICHA del file FICHAS está definido con 80 caracteres, mientras que el campo de WORKING-STORAGE FICHA-W-S tiene una longitud de sólo 17 caracteres).

```
PROCEDURE DIVISION.
LEE FILE.
OPEN INPUT FICHAS.
READ FICHAS INTO FICHA W-S
                AT END... .
```

Si el contenido de la ficha leída (presente en el buffer de entrada) es por ejemplo

AA101283XZ7411CABLETELEFONICO

en el campo FICHA-W-S se tiene

AA101283XZ7411CAB

Si el campo FICHA-W-S hubiese sido, por ejemplo,

```
01 FICHA-W-S                PIC X(150).
```

la instrucción

```
READ FICHAS INTO FICHA-W-S
          AT END... .
```

debería transferir todo el contenido de la ficha leída, llenando las posiciones de la 81ª a la 150ª con caracteres blancos.
La regla citada es válida también para la instrucción WRITE:

- Si la longitud del campo especificado en la cláusula FROM es inferior a la de la descripción del record de salida, este último contendrá todo el campo de WORKING-STORAGE más una serie de espacios en blanco hasta el llenado del record.
- En cambio, en el caso en que el campo de la cláusula FROM tenga una longitud superior a la indicada en la descripción del record, su contenido se truncará a las dimensiones máximas del record de salida.

A pesar de que los formatos que operan directamente en los buffers de entrada/salida son más eficientes porque no prevén ningún transporte de datos de un punto a otro de la memoria, imponen una gestión más corta de las instrucciones. Sin embargo es necesario conocer perfectamente las situaciones en que el contenido del record en curso es fiable y en las que ya no lo es.
A este propósito indicamos una serie de instruc-

ciones erróneas, que ilustran los problemas a los que se ha hecho referencia.

Primer caso:
WRITE RECORD-A.
MOVE RECORD-A TO DISPONIBLE-REC.
Cada operación de escritura deja inutilizable el contenido del record en curso. Análogamente, cada READ deja disponible el nuevo record e inutilizable el anterior.

Segundo caso:
MOVE DISPONIBLE-REC TO RECORD-A.
OPEN INPUT FILE-A.
La operación de apertura de un file no deja disponible el área del record, que por otra parte ya no contiene el valor de DISPONIBLE-REC.

Tercer caso:
```
READ FILE-A
        AT END
        MOVE RECORD-A TO DISPONIBLE-
        REC.
```
La condición de fin de file deja inalcanzable el valor del record en curso.

Escritura en files de impresión. La modalidad de utilización de la instrucción WRITE para los files en disco continúan válidas también para los files asignados a la impresora.
Para estos últimos hay disponibles cláusulas con las que puede controlarse el carro de la impresora posicionándolo según las exigencias del documento deseado.
En cualquier caso recuérdese que los files asignados a la impresora (PRINTER) deben declararse siempre con la cláusula LABEL RECORD OMITED, y que la longitud del record (línea de impresión) debe estar constituida por la longitud útil (máx. 132 caracteres) más uno, puesto que una posición se reserva al carácter de control del carro. Por tanto, el formato completo de la instrucción WRITE es el indicado en la tabla.
Raramente el documento producido por un programa Cobol consiste en una simple lista de informaciones, sino que tiene precisamente la necesidad de presentar los resultados del proceso en listados bien organizados y de fácil lectura. Además, el listado producido por un programa también puede ser muy largo, por lo que es necesario que no sólo la primera página, sino cada página del documento contenga una o más filas de cabecera que aclaren la naturaleza y el significado de los datos indicados.
A continuación se examinan las cláusulas sencillas de la WRITE que permiten la gestión de los files de impresión.

Cláusulas para el salto de líneas. Si no se ha especificado ninguna cláusula en la instrucción WRITE, ésta procede a imprimir el contenido del record en la línea inmediatamente siguiente a la última impresa. En el caso de que el programador quiera alterar esta secuencia de impresión puede declarar al Compilador el número de líneas que debe saltar antes de imprimir la línea todavía en memoria.
En otras palabras

```
WRITE LINEA FROM LINEA-W-S
AFTER ADVANCING 3 LINES.
```

equivale a decir: «después de saltar 3 líneas (AFTER ADVANCING 3 LINES) escribe (WRITE) el record LINEA tomando el contenido del campo de WORKING-STORAGE LINEA-W-S».
Por tanto, es posible obtener antes la impresión de la línea deseada y sólo después el salto de un cierto número de líneas utilizando la forma

```
WRITE LINEA FROM LINEA-W-S
AFTER ADVANCING 3 LINES.
```

FORMATO DE LA INSTRUCCION WRITE

```
WRITE nombre-de-record [FROM nombre-de-dato]

[ { BEFORE }             { nombre-de-dato-1   LINES } ]
[ {        }  ADVANCING  { número-entero      LINES } ]
[ { AFTER  }             { PAGE                     } ]

[      { END-OF-PAGE }                      ]
[ AT   {             }   frase-imperativa   ]
[      { EOP         }                      ]
```

o bien: «escribe (WRITE) el record LINEA tomando el contenido del campo LINEA-W-S antes de avanzar 3 líneas (BEFORE ADVANCING 3 LINES)».

El número de líneas que debe hacerse avanzar el carro (en realidad es el papel que avanza) también puede ser un número entero sin signo contenido en un campo elemental numérico definido en WORKING-STORAGE SECTION.

Por tanto, los ejemplos analizados son equivalentes a los indicados a continuación

```
WRITE LINEA FROM LINEA-W-S
AFTER ADVANCING SALTO LINES.
WRITE LINEA FROM LINEA-W-S
BEFORE ADVANCING SALTO LINES.
```

donde SALTO está decidido en WORKING-STORAGE SECTION en el modo

```
01 SALTO PIC 9(2) VALUE 3.
```

Independientemente del método utilizado para declarar las líneas a saltar, éstas no pueden ser más de 99.

Obsérvese finalmente que las palabras ADVANCING y LINES no son obligatorias y, por tanto, las mismas instrucciones pueden escribirse

```
WRITE LINEA FROM LINEA W-S AFTER 3.
WRITE LINEA FROM LINEA-W-S
AFTER SALTO.
WRITE LINEA FROM LINEA-W-S
BEFORE 3.
WRITE LINEA FROM LINEA-W-S
BEFORE SALTO.
```

Cláusulas para el salto de páginas. De forma análoga a la descrita para el salto de líneas, la instrucción WRITE prevé también una gestión de la página sobre la que efectuar la impresión de la línea en escritura. La clásula PAGE sólo permite saltar a la página inmediatamente siguiente a la que se utiliza:

```
WRITE LINEA FROM LINEA-W-S
AFTER PAGE.
```

equivale a: «después (AFTER) de haberte posicionado en la primera línea a escribir de la nueva página (PAGE) escribe (WRITE) LINEA utilizando el campo LINEA-W-S»;

```
WRITE LINEA FROM LINEA-W-S
BEFORE PAGE.
```

significa: «antes (BEFORE) escribe LINEA usando el contenido de LINEA-W-S y después salta a la primera línea útil de la nueva página (PAGE)».

Hasta ahora se ha hablado genéricamente de «primera línea a escribir de la página»: si el programador no la altera explícitamente con la adecuada cláusula, esta línea la gestiona directamente el sistema operativo.

La cláusula LINAGE. Para poder organizar según las propias exigencias de impresión las páginas de un documento, el programador puede utilizar la cláusula LINAGE en la FILE SECTION. Esta clásula permite declarar al Compilador:

- El número total de líneas a imprimir en el campo de la página (LINAGE); debe ser mayor que 0
- El número de líneas vacías que deben preceder a las líneas a imprimir (TOP); puede ser igual a 0
- El número de líneas vacías que deben seguir a las líneas a imprimir (BOTTOM); puede ser igual a 0
- El número de la línea en que imprimir el record en curso antes de saltar a una nueva página (FOOTING); debe caer entre las filas a imprimir.

El formato completo de la cláusula LINAGE es el siguiente:

FORMATO DE LA CLAUSULA LINAGE

LINAGE IS número-líneas-a imprimir LINES
[WITH FOOTING AT línea-de-footing]
[LINES AT TOP margen-superior]
[LINES AT BOTTOM margen-inferior]

Todos los parámetros declarados deben ser números enteros o nombres de datos numéricos elementales sin signo.

En otras palabras, escribir

```
DATA DIVISION.
FILE SECTION.
FD IMPRIME
LABEL RECORD OMITTED
LINAGE IS 56 LINES
        LINES AT TOP 6
        LINES AT BOTTOM 4
        WITH FOOTING AT 56.
01 LINEA                        PIC X(133).
```

es completamente equivalente a escribir

```
DATA DIVISION.

FILE SECTION.
FD IMPRIME
LABEL RECORD OMITTED

LINAGE IS NUMERO-LINEAS LINES
    LINES AT TOP MARGEN-SUPERIOR
    LINES AT BOTTOM MARGEN-INFERIOR
    WITH FOOTING LINEA-FOOTING.
01 LINEA                      PIC X(133).

WORKING-STORAGE SECTION.
01 NUMERO-LINEAS         PIC 9(2) VALUE 56.
01 MARGEN-SUPERIOR          PIC 9 VALUE 6.
01 MARGEN-INFERIOR          PIC 9 VALUE 4.
01 LINEA-FOOTING         PIC 9(2) VALUE 56.
```

Obsérvese que todas las cláusulas de la LINAGE son opcionales, y en el caso en que no se utilicen, automáticamente se asumen (por omisión) los siguientes valores:

```
línea-de-footing    = número líneas a imprimir
margen-superior     = 0
margen-inferior     = 0
```

Además de las posibilidades descritas, la cláusula LINAGE puede agilizar notablemente la gestión de la impresión. Efectivamente, en el momento en que se abre un file de impresión al cual hay asociada la cláusula LINAGE, también se asigna un registro especial (LINAGE-COUNTER), gestionado automáticamente por el Cobol durante la ejecución de las instrucciones WRITE. En este registro hay disponible el número de la fila sobre la que está posicionada la impresora en el ámbito de una misma página.
El significado de todo lo dicho se evidencia en el ejemplo indicado abajo.
Del ejemplo se deduce que, cuando se utiliza la cláusula LINAGE en la descripción de un file de impresión, el Cobol puede «sentir», mediante el LINAGE-COUNTER, el final de la página delimitada por los parámetros en FILE-SECTION.

IDENTIFICATION DIVISION

```
–
FILE SECTION.
FD IMPRIME
  LABEL RECORD OMITTED
  LINAGE IS 50
          TOP 10
          BOTTOM 6.
01 LINEA                       PIC X(133).
WORKING-STORAGE SECTION.
01 LINEA-W-S.
  05 ..........
  –
  –
PROCEDURE DIVISION.
ABRE-FILE.
  OPEN OUTPUT IMPRIME
  –
  –
  –
  –
  –
WRITE LINEA FROM LINEA-W-S.
  –
  –
WRITE LINEA FROM LINEA-W-S AFTER 3.
  –
WRITE LINEA FROM LINEA-W-S.
  –
  –
  –
  –
  –
  –
WRITE LINEA FROM LINEA-W-S AFTER PAGE
```

- OPEN OUTPUT IMPRIME: Asigna el file y pone a 0 el contenido del LINAGE-COUNTER.
- WRITE LINEA FROM LINEA-W-S.: Imprime el contenido de LINEA-W-S en la undécima línea de la hoja, o bien después de un margen superior a 10 líneas (TOP 10). El LINAGE-COUNTER es incrementado en 1.
- WRITE LINEA FROM LINEA-W-S AFTER 3.: Hace avanzar 3 líneas el papel, imprime el contenido de LINEA-W-S e incrementa en 3 el LINAGE-COUNTER.
- WRITE LINEA FROM LINEA-W-S.: Supóngase que, con las instrucciones WRITE utilizadas hasta ahora, el carro esté posicionado en la 50ª línea, o bien en la última línea a escribir, de acuerdo con la cláusula LINAGE. En este caso, el contenido de LINEA se imprime en la línea utilizable de la siguiente página, y el LINAGE-COUNTER se coloca en 1, para tener en cuenta sólo las líneas impresas en la página acabada de empezar.
- WRITE LINEA FROM LINEA-W-S AFTER PAGE: El comportamiento es análogo al descrito por la WRITE anterior, pero lo comanda el programador si en este punto del programa desea cambiar de página.

Aprovechando esta capacidad es posible declarar en la instrucción WRITE una acción que el programa debe realizar inmediatamente después del fin de una página.
Supóngase que se quiere imprimir una cabecera al principio de cada página. En este caso, si las instrucciones utilizadas para la impresión de la cabecera se reúnen todas en una SECTION de nombre CABECERA, puede escribirse

```
WRITE LINEA FROM LINEA-W-S
AT END-OF-PAGE PERFORM
CABECERA.
```

o más sintéticamente

```
WRITE LINEA FROM LINEA-W-S
EOP PERFORM CABECERA.
```

Las instrucciones ACCEPT y DISPLAY

Estos dos verbos de I/O son comparables respectivamente a una READ y a una WRITE, pero están orientados esencialmente a pequeñas cantidades de datos.
Mediante el verbo ACCEPT, el programador tiene la posibilidad de detener el proceso para tomar datos de la consola del sistema, del lector de fichas o de algunos registros particulares.
Contrariamente a lo que sucede para la instrucción READ, la ACCEPT no necesita la asignación de ningún file, ni en el interior ni en el exterior del programa.
La instrucción tiene el formato indicado abajo, y permite transferir en el campo de WORKING-STORAGE nombre-de-dato la entrada procedente de (FROM) una de las áreas de sistema encerradas entre paréntesis.
Así, el programa en ejecución puede adquirir parámetros variables del exterior. Estos parámetros tienen esencialmente las funciones

- de documentación
- de parametrización del proceso
- de diálogo mediante la consola.

El significado de esta subdivisión se aclarará con el ejemplo indicado después de la descripción del verbo DISPLAY.
Por el momento, téngase presente que en el campo nombre-de-dato se transfieren

1 / la respuesta tecleada en consola por el operador, cuando hay especificada la cláusula FROM CONSOLE
2 / el contenido de una ficha, cuando se emplea la cláusula FROM CARD-READER
3 / la fecha del día en el formato AAMMGG cuando se usa la cláusula FROM DATE
4 / la fecha del día en el formato AAGGG cuando se usa la cláusula FROM DAY
5 / la hora del día expresada en 8 caracteres en la forma HHPPSSCC cuando se utiliza la cláusula FROM TIME

Para aclarar las notaciones usadas en los últimos 3 formatos, supóngase que el programa se realiza el día 15 de enero de 1984 cuando han transcurrido 8 horas, 12 minutos, 35 segundos y 75 centésimas de segundo a partir de la medianoche. Las tres instrucciones siguientes

```
ACCEPT FECHA-OFICIAL     FROM DATE.
ACCEPT FECHA-JULIANA     FROM DAY.
ACCEPT HORA              FROM TIME.
```

implantan los respectivos campos de llegada a los valores que hay a su lado:

```
FECHA-OFICIAL   = 840115
FECHA-JULIANA   = 84015
HORA            = 08123575
```

Los campos de llegada deben haberse definido respectivamente con longitudes de 6, 5 y 8 caracteres, numéricos y sin signo.

El verbo DISPLAY permite enviar un mensaje a la consola del sistema o a la impresora.

FORMATO DE LA INSTRUCCION ACCEPT

```
                                { CONSOLE     }
                                { CARD-READER }
ACCEPT nombre-de-dato FROM      { DATE        }
                                { DAY         }
                                { TIME        }
```

Este mensaje puede estar compuesto indiferentemente de constantes alfanuméricas o del contenido de campos descritos en WORKING-STORAGE. En síntesis, el formato general de la instrucción es el indicado en la tabla.

Un ejemplo de empleo de las instrucciones ACCEPT y DISPLAY se ve en la tabla de abajo, correspondiente a un simple programa que debe

FORMATO DE LA INSTRUCCION DISPLAY

```
          {nombre-de-dato-1}  [{nombre-de-dato-2}]            {CONSOLE}
DISPLAY   {                }  [{                }] ..... UPON {       }
          {constante-1     }  [{constante-2     }]            {PRINTER}
```

EJEMPLO DE USO DE LAS INSTRUCCIONES ACCEPT Y DISPLAY

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ACCEPT-DISPLAY.
REMARKS. ESTE PROGRAMA ES UN EJEMPLO DE USO
        DE LAS INSTRUCCIONES ACCEPT Y DISPLAY.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER..................
OBJECT-COMPUTER..................
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MOVIMIENTOS ASSIGN TO DISC MOVIMIENTOS.
    SELECT ESTADISTICAS ASSIGN TO DISC ESTADISTICAS.
DATA DIVISION.
FILE SECTION.
FD MOVIMIENTOS
  LABEL RECORD STANDARD.
01   REC-MOV.                          PIC X(14).
FD ESTADISTICAS
  LABEL RECORD STANDARD.
01   REC-STA                           PIC X(10).
*
WORKING-STORAGE SECTION.
01.  REC-MOV-WS.
     05 ART-MOV.                       PIC 9(3).
     05 FECHA-MOV.
        10 AÑO-MOV.                    PIC 9(2).
        10 MES-MOV.                    PIC 9(2).
        10 DIA-MOV.                    PIC 9(2).
     05 CANTIDAD                       PIC 9(5).
*
01   REC-STA-WS.
     05 ART-STA                        PIC 9(3).
     05 MES-STA                        PIC 9(2).
     05 VENDIDO                        PIC 9(5).
*
01   FICHA-PARAMETRO.
     05 TIPO-SK-PAR                    PIC XX.
     05 ART-SK-PAR                     PIC 9(3).
*
01    PETICION MES                     PIC X(80)   VALUE
     ' TECLEAR NUMERO MES A PROCESAR'.
*
01    MES-DE-CONSOLA                   PIC 9(2).
*
01   FECHA-PROCESO                     PIC 9(6).
01   FECHA-PROC REDEFINES FECHA-PROC.
     05    AÑO-PROC                    PIC 9(2).
     05    MES-PROC                    PIC 9(2).
     05    DIA-PROC                    PIC 9(2).
*
01   HORA-PROCESO                      PIC 9(8).
01   HORA-PROC   REDEFINES HORA-PROCESO.
     05    HORAS                       PIC 9(2).
     05    MINUTOS                     PIC 9(2).
```

```
       05 SEGUNDOS                   PIC 9(2).
       05 CENTESIMAS                 PIC 9(2).
*
 01  PROCESO.
       05 FILLER                     PIC X(17) VALUE
       '  PROCESO DEL'.
       05 DIA-DISPLAY                PIC 99.
       05 FILLER                     PIC X     VALUE '/'.
       05 MES-DISPLAY                PIC 99.
       05 FILLER                     PIC X     VALUE '/'.
       05 AÑO-DISPLAY                PIC 99.
       05 FILLER                     PIC X(10) VALUE
             A LAS'.
       05 HORAS-DISPLAY              PIC 99.
       05 FILLER                     PIC X     VALUE ':'.
       05 MINUTOS-DISPLAY            PIC 99.
       05 FILLER                     PIC X     VALUE ':'.
       05 SEGUNDOS-DISPLAY           PIC 99.
       05 FILLER                     PIC X     VALUE ':'.
       05 CENTESIMAS-DISPLAY         PIC 99.
*
 01  EXTRACCION.
       05 FILLER                     PIC X(42) VALUE.
          'EXTRACCION Y ACOMODO PARA EL ARTICULO '.
       05 ART-DISPLAY                PIC 9(3).
 PROCEDURE DIVISION.
 INICIO SECTION.
 ACEPTA.
       ACCEPT FECHA-PROCESO FROM DATE.
       ACCEPT  HORA-PROCESO FROM TIME.
       DISPLAY PETICION-MES
         UPON CONSOLA.
       ACCEPT MES-DE-CONSOLA FROM CONSOLA.
 DETERMINA-OPEN.
       IF MES-DE-CONSOLA = 1
          OPEN OUTPUT ESTADISTICAS
       ELSE
          OPEN EXTEND ESTADISTICAS.
 ACEPTA-SK-PAR.
     ACCEPT FICHA-PARAMETRO FROM CARD-READER.
     OPEN INPUT MOVIMIENTOS.
 LEE-MOVIMIENTOS.
     LEE MOVIMIENTOS INTO REC-MOV-WS
         AT END
         PERFORM CIERRE
          STOP RUN.
     IF ART-MOV NOT = ART-SK-PAR
        GO TO LEE MOVIMIENTOS.
     PERFORM PROCESA-RECORDS.
     WRITE REC-STA FROM REC-STA-WS.
     GO TO LEE-MOVIMIENTOS.
*
 PROCESA-RECORD SECTION.
 PROC-REC.
     .................
 PROC-REC-EX. EXIT.
*
 CIERRE SECTION.
 CIERRA.
     CLOSE MOVIMIENTOS
     CLOSE ESTADISTICAS.
     MOVE ART-SK-PAR TO ART-DISPLAY.
     MOVE AÑO-PROC    TO AÑO-DISPLAY.
     MOVE MES-PROC    TO MES-DISPLAY.
     MOVE DIA-PROC    TO DIA DISPLAY.
     MOVE HORAS       TO HORAS-DISPLAY.
     MOVE MINUTOS     TO MINUTOS-DISPLAY.
     MOVE SEGUNDOS    TO SEGUNDOS-DISPLAY.
     MOVE CENTESIMAS TO CENTESIMAS-DISPLAY.
     DISPLAY PROCESO      UPON PRINTER.
     DISPLAY EXTRACCION  UPON PRINTER.
     DISPLAY '** FIN PROCESO **'
            UPON PRINTER.
 CIERRE-EX. EXIT.
*
```

1 / leer un file en disco en el que hay memorizados los movimientos mensuales de todos los artículos vendidos en un almacén

2 / extraer sólo los datos de un artículo, cuyo código está perforado en una ficha

3 / poner los datos procesados a continuación de los de un file de estadísticas residentes en disco

4 / abrir oportunamente el file ESTADISTICAS (crearlo o abrirlo en extensión) y pedir al operador (de consola) si los datos en proceso corresponden a enero o a otro mes cualquiera del año

5 / finalizado el proceso debe escribirse en impresora un mensaje formulado como sigue:

PROCESO DEL (fecha) A LAS (hora)
EXTRACCION Y ACOMODAMIENTO PARA
EL ARTICULO (código-artículo)
FIN PROCESO.

Abajo y en la pág. 1010 se han indicado el flujo de datos y el diagrama de flujo del programa.

Verbos aritméticos

Los verbos aritméticos permiten trabajar sobre datos contenidos en los campos mnemónicos para calcular el valor de determinadas expresiones aritméticas a las cuales concurren dichos datos.

Un lenguaje de programación debe poder gestionar ágilmente las informaciones del ambiente al que está orientado. Por tanto, dado su destino, el Cobol está dotado de un limitado conjunto de operaciones matemáticas, suficiente para satisfacer las exigencias de cálculo del ambiente en que se desarrolla: el comercial.

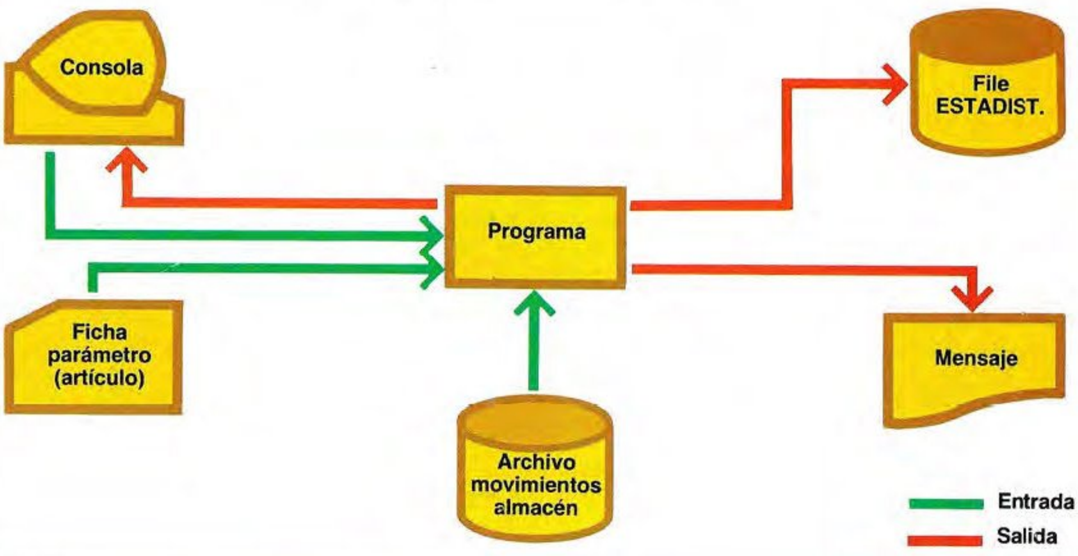
Antes de proceder al examen de las instrucciones aritméticas es indispensable conocer la forma en que los datos numéricos se representan en la memoria del procesador. Por tanto, describiremos algunas cláusulas gracias a las cuales es posible imponer un tipo de representación en lugar de otro, de manera que se tengan los elementos que permitirán estructurar el proceso de los datos numéricos de la forma más adecuada.

Representación de los datos en memoria: la cláusula USAGE

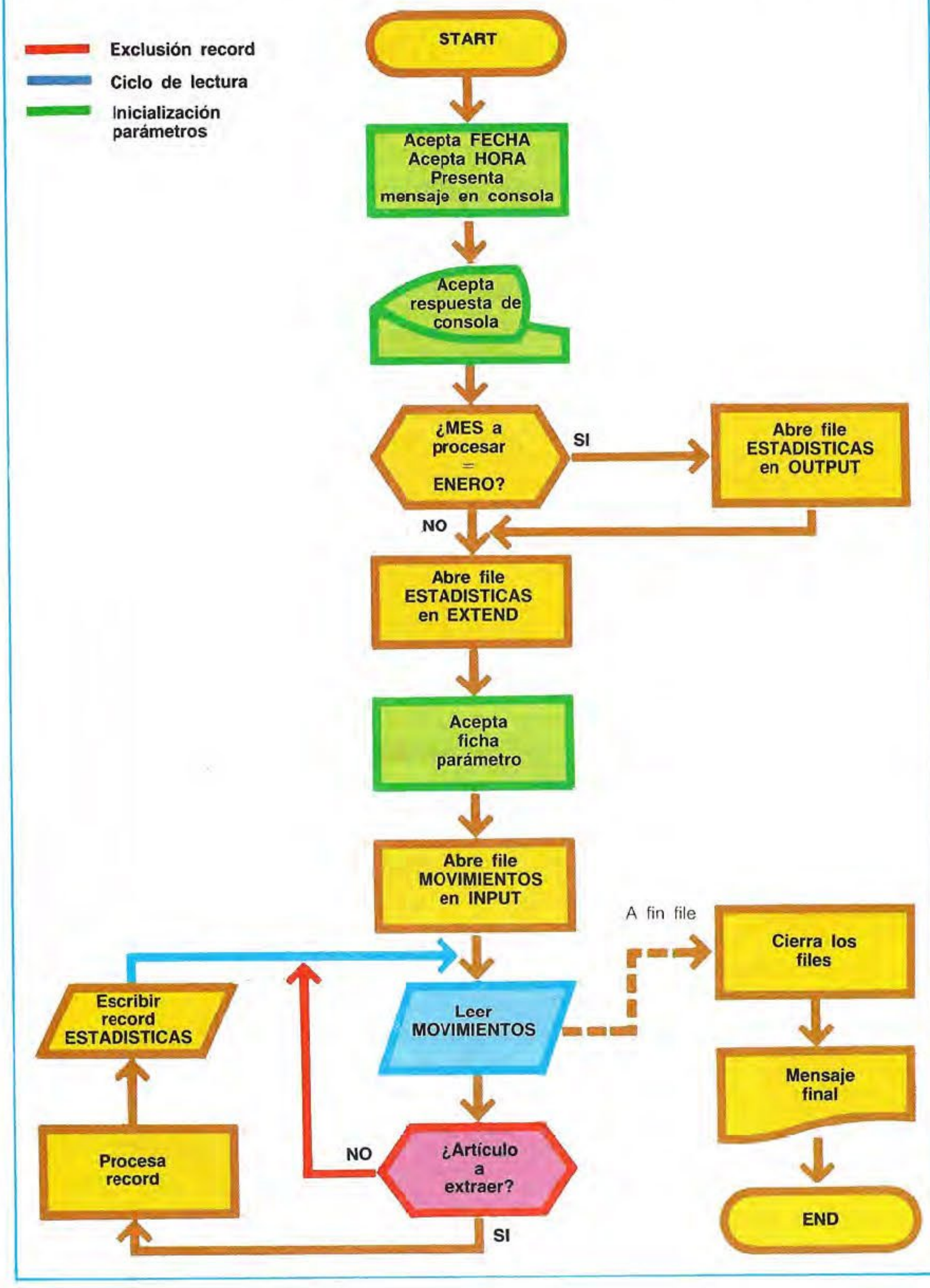
Durante las operaciones de transferencia de datos (MOVE) o aritméticas, el calculador efectúa conversiones de representación, de manera que pueda actuar sobre los datos de forma homogénea.

Una apropiada representación interna de los datos que corresponden a ciertos artículos, descarga el sistema de las operaciones de con-

EJEMPLO DE USO DE LAS INSTRUCCIONES ACCEPT Y DISPLAY FLUJO DE LOS DATOS



EJEMPLO DE EMPLEO DE LAS INSTRUCCIONES ACCEPT Y DISPLAY



versión, con notable beneficio en términos de velocidad de ejecución. Para los datos numéricos, en ANS COBOL se han previsto dos tipos de representación interna:

DISPLAY (codificación hexadecimal)

COMPUTATIONAL (codificación binaria)

Para definir un dato cuya representación interna debe tenerse en codificación binaria, por ejemplo puede escribirse indiferentemente

```
01 CAMPO    PIC S9(5)
            USAGE IS COMPUTATIONAL.
```

o también

```
01 CAMPO    PIC S9(5) COMP.
```

Si la cláusula USAGE se omite, el Compilador asume por omisión la representación interna de tipo DISPLAY. Esto significa que, para el Compilador, las tres descripciones siguientes son completamente equivalentes:

```
01 CAMPO    PIC S9(5).
01 CAMPO    PIC S9(5)
            USAGE IS DISPLAY.
01 CAMPO    PIC S9(5) DISPLAY.
```

USAGE DISPLAY. Con este tipo de representación, cada cifra ocupa un carácter (byte) del campo en memoria.
Por ejemplo, en el caso de la descripción

```
01 CAMPO    PIC 9(4) VALUE 6.
```

no se ha especificado la cláusula USAGE y el Compilador asume DISPLAY, y por ser el campo inicializado en el valor 6, tiene como representación interna 0006.

USAGE COMPUTATIONAL. La cláusula USAGE COMPUTATIONAL (COMP) declara al Compilador que el contenido del campo en examen debe tener una representación de tipo binario. Es evidente que el uso del código binario no limita el Compilador a operaciones de conversión, puesto que el calculador opera según este tipo de representación.
El uso de la cláusula USAGE COMP, además de permitir tiempos de proceso más cortos, comporta también una menor ocupación de memoria. Efectivamente, mientras un campo de USAGE DISPLAY ocupa en memoria tantos bytes como cuantos son los caracteres declarados en la PICTURE, con la USAGE COMP, la ocupación en memoria es inferior, a pesar de que permite la representación de los mismos números. Considérese por ejemplo que deba definirse un campo que pueda contener hasta 10 cifras. Si su descripción fuese

```
01 CAMPO    PIC S9(10).
```

el Compilador reservaría 10 bytes en memoria. El mismo campo descrito como

```
01 CAMPO    PIC S9(10)
            USAGE COMPUTATIONAL.
```

ocuparía en memoria sólo 8 bytes.
La ventaja todavía es más significativa para campos de longitud superior a 10 cifras. Si se considera el caso de un campo de dimensiones máximas previstas por el Cobol, o sea de 18 cifras, descrito en el modo

```
01 CAMPO    PIC S9(18) COMP.
```

su ocupación de memoria todavía será igual a 8 bytes.
Finalmente puede trazarse la segunda tabla, en la que se indica la ocupación real de memoria para campos COMPUTATIONAL en función de diversos intervalos de longitud declarados en la cláusula PICTURE.

Número de cifras (9) en la PICTURE	Ocupación de memoria
de 1 a 4 de 5 a 9 de 10 a 18	2 bytes 4 bytes 8 bytes

Para comprender más a fondo la distinción real que existe entre las dos representaciones analizadas hasta ahora, considérense dos campos, CAMPO-1 y CAMPO-2, declarados respectivamente DISPLAY y COMPUTATIONAL:

```
01 CAMPO-1  PIC S9(2).
01 CAMPO-2  PIC S9(2) COMP.
```

Debe observarse que los dos ocupan en memoria dos bytes: CAMPO-1 porque esto es expre-

samente declarado por la cláusula PICTURE, y CAMPO-2 en base a la tabla de equivalencia indicada anteriormente.
Si se supone que tanto uno como otro campo contienen el número 17, la representación en memoria correspondiente a los dos campos es la siguiente:

CAMPO-1

(F) (1) (F) (7)

| 1111 , 0001 | 1111 , 0111 |

(codificación hexadecimal)

CAMPO-2

(2^4) (2^0)

| 0000 , 0000 | 0001 , 0001 |

(codificación binaria)

Como puede observarse, el contenido de CAMPO-2 es el código binario normal del número 17 con una extensión de 16 bits.
También se recuerda que el primer bit de la izquierda del campo está reservado para el signo. Si este bit se pone a 0, el número es positivo (ver ejemplo), mientras los números negativos se representan con la combinación de bits obtenida del complemento a 2. Considérese el caso del campo descrito a continuación

01 CAMPO-3 PIC S9(4) COMP.

que en un cierto momento su contenido valga, por ejemplo, – 1234.
Si el número contenido fuese + 1234, su representación interna sería la siguiente:

| 0000 , 0100 | 1101 , 0010 |

Como puede observarse, tratándose de un número positivo, el primer bit es igual a 0.
En realidad, como debe memorizarse el número – 1234, la máquina procede a representar el mismo como complemento a 2 de la configuración anterior, procediendo según los pasos descritos a continuación

1 / Representación del número + 1234

← 1 byte →

| 0000 , 0100 | 1101 , 0010 |

2 / Complemento a 1

| 1111 , 1011 | 0010 , 1101 |

3 / Suma de 1 +1

4 / Representación del número – 1234

| 1111 , 1011 | 0010 , 1110 |

Muchos Compiladores prevén el uso de otros tipos de representación para los datos numéricos, no previstos por el Cobol estándar. Estas representaciones se declaran con los siguientes formatos de la cláusula USAGE:

USAGE IS COMPUTATIONAL-3.
USAGE IS COMPUTATIONAL-1.
USAGE IS COMPUTATIONAL-2.

Los mismos formatos son equivalentes a

COMP-3.
COMP-1.
COMP-2.

USAGE COMP-3. Esta representación, llamada también **empaquetada** (packed), reserva un byte cada dos cifras del campo descrito, a excepción del último byte de la derecha, en que se memoriza una sola cifra más la ubicación del signo. Por ejemplo, considérese el campo CAMPO-4 cuya descripción sea

01 CAMPO-4 PIC 9(3) COMP-3.

y cuyo valor en un instante dado sea 456.
Como con USAGE COMP-3 cada cifra ocupa un semibyte, y el último está reservado para el signo, se tendrá esta representación interna

← 1 byte → ← 1 byte →

| 4 , 5 | 6 , F |

en la que la letra F tiene en cuenta el hecho de que CAMPO-4 no prevé el signo (o bien que falta el símbolo S en la PICTURE).
Debe observarse que la asignación en memoria de los bytes necesarios para representar el número de cifras que necesita la PICTURE siempre se hace de manera que ocupe un número entero de bytes. Es decir, un campo declarado de dos cifras con signo [PIC S9(2)] necesitaría

el uso de tres semibytes, dos para las cifras más uno para el signo. En este caso, el Compilador asigna también el cuarto semibyte, expandiendo con un 0 a la izquierda la parte no utilizada. Consideremos, por ejemplo, el campo descrito como sigue:

01 CAMPO-5 PIC S9(2) COMP-3.

y supongamos que CAMPO-5 contiene en un cierto momento el valor + 12.
Su representación interna será

0	1	2	C
← 1.er byte →		← 2.º byte →	

La letra C declara que el valor contenido en el campo es positivo.
Para completar la descripción de la cláusula COMP-3 considérese el campo

01 CAMPO-6 PIC S9(4)V9(2) COMP-3.

La ocupación de memoria es de 6 semibytes (tantos como las cifras declaradas por la PICTURE) más un semibyte para la codificación del signo, más un semibyte añadido por el Compilador para utilizar un número entero de bytes; en total 8 semibytes, o bien 4 bytes. Si un valor momentáneo de CAMPO-6 es – 4867.15, su representación es

0	4	8	6	7	1	5	D

en la que la letra D tiene en cuenta el signo negativo del número memorizado.

USAGE COMP-1 y COMP-2. Estos dos tipos de representación interna, llamadas también **floating point** (coma flotante), se usan escasamente en el ámbito del Cobol. Su uso está justificado cuando deben manipularse números muy grandes, pero comporta una precisión menor con respecto a otros tipos de representación. Mediante estas representaciones internas pueden manipularse números que van de 10^{-78} a 10^{+75} con un grado de precisión que se extiende hasta la 6.ª cifra decimal para el COMP-1, y hasta la 16.ª para el COMP-2.
Esto también justifica el limitado uso en el ambiente comercial en que se mueve el Cobol.
El formato del USAGE COMP-1 o COMP-2 difiere del de las representaciones descritas hasta ahora. Efectivamente, si un campo se declara en coma flotante, el Compilador le asigna automáticamente una palabra (COMP-1) o dos palabras (COMP-2).
En consecuencia, un campo declarado en coma flotante de simple precisión se describe por

01 CAMPO-7 COMP-1.

mientras que uno de doble precisión puede ser, por ejemplo,

01 CAMPO-8 COMP-2.

La representación interna de un número en coma flotante consta de dos partes: una parte representa las cifras significativas del número (**mantisa**) y la otra representa el exponente que determina el valor real del número (**característica**). Sólo los modernos calculadores están dotados de un circuito hardware especial que puede efectuar las operaciones de normalización necesarias para este tipo de representación; por tanto, nos limitaremos a exponer las disposiciones de las partes indicadas en las dos representaciones COMP-1 y COMP-2 en una máquina con palabras de 32 bits o menos:

COMP-1	S	Característica	Mantisa
Bit n.	0	1 7	8 31
COMP-2	S	Característica	Mantisa
Bit. n.	0	1 7	8 63

El buen conocimiento de las representaciones internas de los datos numéricos constituye una buena base para el planteo de un programa rápido y fiable.
Para completar las informaciones relativas a este tema, téngase presente que el Cobol permite, en el ámbito de una misma operación, el uso de campos numéricos con representaciones internas diferentes. Esto es posible, como ya se ha indicado, gracias a procedimientos de conversión gestionados directamente por el Compilador. En cualquier caso, el proceso de conversión está vinculado a una jerarquía que establece un orden de prioridad. La jerarquía citada, en orden decreciente de prioridad, es la que a continuación se indica:



El microprocesador aplicado al control de una caja registradora.

COMP-2
COMP-1
COMP-3
DISPLAY
COMP

Todo lo indicado quedará más claro cuando se hable de los verbos de tipo aritmético.

Las expresiones aritméticas

Una expresión aritmética, cosa bien conocida, permite efectuar una serie de operaciones, a veces complejas, sobre operandos y constantes; el resultado puede asociarse a una variable. Por ejemplo,

$$A = (B + C)^2 + \frac{D}{(E + F)} - (G + H)^P + 100$$

constituye una expresión en que, a la izquierda del signo igual, aparece la variable dependiente A, cuyo valor es igual al resultado que se deriva del proceso del segundo miembro (cuando cada una de las variables del mismo tenga un valor definido).

En el Cobol, el concepto de expresión aritmética no es diferente del clásico, y adopta completamente sus reglas.
Así, el programador que quiere calcular el valor de una expresión aritmética tiene a su disposición las cinco operaciones fundamentales de la matemática del calculador, más los signos de paréntesis con que puede alterar la jerarquía de ejecución de ciertas operaciones.
En el Cobol, las operaciones de la matemática están representadas por los operadores

+	Suma
–	resta
*	multiplicación
/	división
**	elevación a potencia.

Con estas notaciones, la expresión anterior puede escribirse así:

A = (B + C)**2 + D/(E + F)–(G + H)**P + 100

Durante el cálculo de una expresión, el Cobol respeta completamente las reglas fundamenta-

les de la aritmética, o sea la jerarquía con que deben realizarsè las diversas operaciones. Esta jerarquía impone en primer lugar la resolución de las elevaciones a potencia, después la de multiplicaciones y divisiones y finalmente la de sumas y restas.
Por ejemplo, el cálculo de la expresión $A = 7 + 5 * 4 - 6 ** 2$ procede según los siguientes pasos:

$$A = 7 + 5 * 4 + 36$$
$$A = 7 + 20 + 36$$
$$A = 63$$

En el caso de que en una misma expresión aparezcan operadores con igual nivel de prioridad, se resuelven en el orden en que se encuentran en la expresión procediendo de izquierda a derecha si se trata de + – * y /, y de derecha a izquierda si se trata de elevaciones a potencia (**).
Por ejemplo, las operaciones presentes en la expresión

$$A = B * C / D + E - F + G ** 2 + H ** 3$$

se realizan en el orden

1 / H ** 3
2 / G ** 2
3 / B * C
4 / (resultado de B * C) / D
5 / (resultado de B * C / D) + E
6 / (resultado de B * C / D + E) – F
7 / ...

Si por exigencias de cálculo es necesario alterar las prioridades descritas, el programador tiene a disposición hasta 9 niveles de paréntesis; mediante los paréntesis puede reagrupar de cualquier modo los datos a tratar. Por ejemplo, utilizando los paréntesis de la manera

$$A = ((B + (C * D)) ** 2 + E / (F - G)) ** 3$$

las agrupaciones más internas se calculan en primer lugar, respetando en su interior las reglas generales ya descritas.

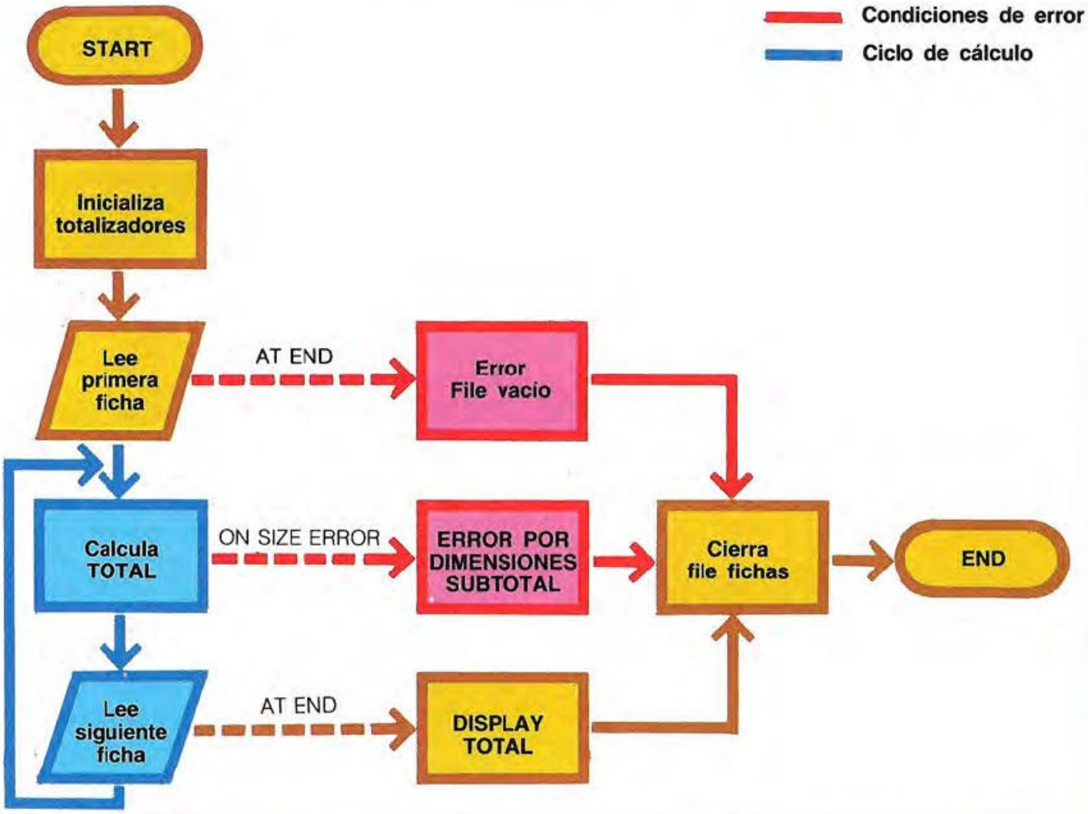
La instrucción COMPUTE

Hasta ahora, a pesar de adoptar la simbología del Cobol, las expresiones se han escrito en la forma clásica de la aritmética.
Para poder obtener el resultado del cálculo de una expresión, el Cobol dispone de una instrucción adecuada. Esta instrucción permite transferir a un campo de datos el valor de una expresión aritmética desde otro campo de datos o desde una constante numérica.
El formato general de la instrucción COMPUTE es el indicado en la tabla de abajo.
Para ilustrar su uso, considérese el siguiente ejemplo.
Debe leerse un file de fichas en el que en cada una de ellas se ha indicado la cantidad comprada de un cierto artículo, su costo unitario, el porcentaje de descuento aplicado y los gastos de expedición por unidad. Para cada ficha leída (o bien por cada compra efectuada) se quiere conocer el costo de la partida. Esto debe contribuir al cálculo de un total de las compras para evidenciarlo en consola. Téngase en cuenta que el total obtenido debe contener también el total de un proceso anterior, tecleado directamente por el operador de la consola cuando el programa se lo pide mediante un adecuado mensaje.
El diagrama de flujo del programa se ha indicado en la pág. 1016, y en la pág. 1017 pueden verse los flujos completos y la ficha perforada. El listado se ha representado en las págs. 1018 y 1019.
Observando el ejemplo puede verse cómo se ha insertado un contador de fichas leídas (CUENTA-FICHAS), no pedido explícitamente por las especificaciones.
Efectivamente, constituye una buena norma prever siempre un contador de los records leídos o escritos, para tener una indicación acerca de los datos movidos por el programa.

FORMATO GENERAL DE LA INSTRUCCION COMPUTE

```
                                          { expresión aritmética }
COMPUTE nombre-de-dato [ROUNDED]          { nombre-de-dato-1     }
                                          { constante numérica   }
                                          [ON SIZE ERROR frase-imperativa].
```

EJEMPLO DE APLICACION DEL VERBO COMPUTE
DIAGRAMA DE FLUJO



En el ejemplo se completan todos los formatos y las cláusulas de la instrucción COMPUTE:

COMPUTE CUENTA-FICHAS = 0

implanta en el contador CUENTA-FICHAS el valor de la constante numérica 0;

COMPUTE TOTAL = TOTAL-ANTERIOR

implanta en el campo TOTAL el valor contenido en el campo TOTAL-ANTERIOR (atención: el signo – es un separador insertado en el nombre del campo).

```
COMPUTE NETO-COMPRA ROUNDED =
    (CANTIDAD * COSTO – UNITARIO) –
    ((CANTIDAD * COSTO – UNITARIO) *
    DESCUENTO / 100) +
    (CANTIDAD * GASTOS-UNITARIOS).
```

Calcula el valor de la expresión a la derecha del signo igual e implanta este valor en el campo NETO-COMPRA respetando la alineación con respecto al punto decimal en la posición prevista por la PICTURE.

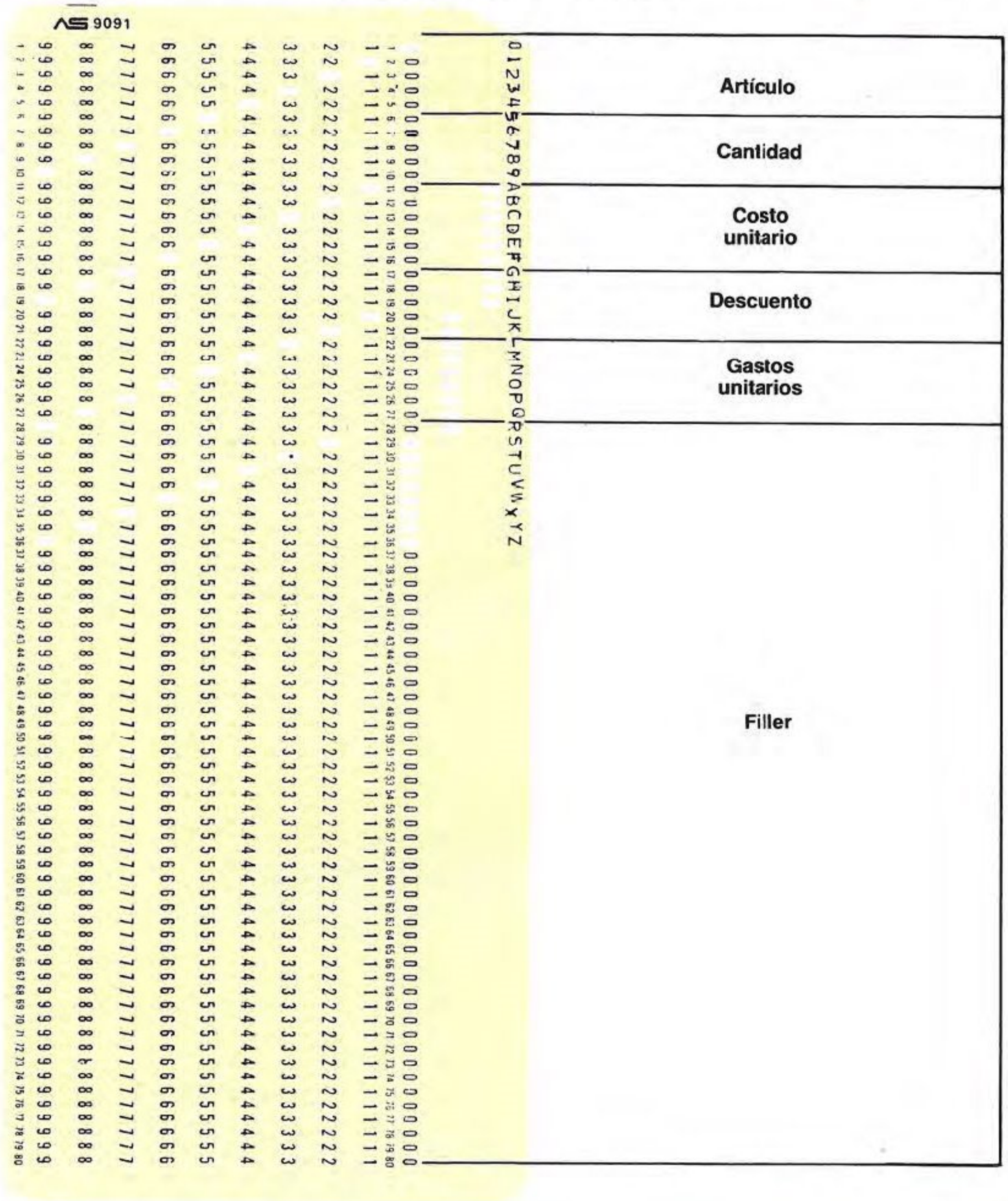
Con este formato, el valor calculado no se trunca simplemente en la tercera cifra decimal; si esta última es superior a 5 se tiene un redondeo al valor decimal inmediatamente superior. Es decir, si el valor de la expresión fuese 785642.3896, la cláusula ROUNDED hace que NETO-COMPRA contenga 785642.390. Y viceversa, si esa cláusula no se hubiese especificado, el contenido del campo habría sido 785642.389.

Sabido es que un contador en cualquier base, después de haber alcanzado el número máximo que puede representar, vuelve a 0 o bien, según el argot habitual, se «desborda». Para acla-

FLUJO DE LOS DATOS



PERFORACION DE FICHA



EJEMPLO DE APLICACION DEL VERBO COMPUTE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALCULO.
REMARKS. ESTE PROGRAMA ES UN EJEMPLO
          DE APLICACION DE LA INSTRUCCION
          COMPUTE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. .............
OBJECT-COMPUTER. .............
IMPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FICHAS ASSIGN TO CARD-READER FICHAS.
DATA-DIVISION.
FILE SECTION.
FD FICHAS
   LABEL RECORD OMITTED.
01 FICHA                           PIC X(80).
*
*
*
WORKING-STORAGE SECTION.
01  FICHA-WS.
    05 ARTICULO                    PIC X(5).
    05 CANTIDAD                    PIC S9(5).
    05 COSTO-UNITARIO              PIC S9(3)V9(3)   COMP.
    05 DESCUENTO                   PIC S9(2)V9(3)   COMP.
    05 GASTOS-UNITARIOS            PIC S9(3)V9(3)   COMP.
    05 FILLER                      PIC X(53).
*
01 TOTAL - ANTERIOR                   PIC S9(15)       COMP.
*
01 TOTAL                           PIC S9(15)       COMP.
*
01 NETO-COMPRA                     PIC S9(8)V9(3)   COMP.
*
01 SUBTOTAL                        PIC S9(15)V9(3)  COMP.
*
01 CUENTA-FICHAS                   PIC S9(5)        COMP.
*
*
*
*
*
PROCEDURE DIVISION.
MAIN SECTION.
INICIALIZA.
    COMPUTE CUENTA-FICHAS = 0
    COMPUTE SUBTOTAL = 0
    DISPLAY ' = TECLEAR EL TOTAL ANTERIOR :
            UPON CONSOLE.
    ACCEPT TOTAL-ANTERIOR FROM CONSOLE.
    COMPUTE TOTAL = TOTAL-ANTERIOR.
*
*
*
ABRE - FILE.
    OPEN INPUT FICHAS.
PRIMERA LECTURA.
    READ FICHAS INTO FICHA-WS
         AT END
         DISPLAY '** FILE FICHAS VACIO **'
                 UPON CONSOLE.
                 PERFORM CIERRE.
CALCULA.
    COMPUTE CUENTA-FICHAS = CUENTA-FICHAS + 1.
    COMPUTE NETO-COMPRA ROUNDED =
                  (CANTIDAD * COSTO-UNITARIO) -
                  ((CANTIDAD * COSTO-UNITARIO) *
                     DESCUENTO / 100) +
```

```
                        (CANTIDAD * GASTOS-UNITARIOS).
     COMPUTE SUBTOTAL = SUBTOTAL + NETO-COMPRA
                             ON SIZE ERROR
                             DISPLAY ' * CAMPO SUBTOTAL INSUFICIENTE
                        'AVISAR AL RESPONSABLE *'
                                      UPON CONSOLE.
                             PERFORM CIERRE.
 OTRAS-LECTURAS.
     READ FICHAS INTO FICHA-WS
                 AT END
                 PERFORM CIERRE.
     GO TO CALCULA.
*
*
*
*
*
 CIERRE SECTION.
 CIERRA-FILE.
     CLOSE FICHAS.
     DISPLAY '** FICHAS LEIDAS = ' CUENTA-FICHAS
             UPON CONSOLE.
     COMPUTE TOTAL ROUNDED = TOTAL + SUBTOTAL.
     DISPLAY ' '
             UPON CONSOLE.
     DISPLAY '** TOTAL ACTUAL = ' TOTAL
             UPON CONSOLE.
     STOP RUN.
*
*
*
```

rar las ideas, piénsese en el cuentakilómetros de los automóviles. Generalmente, estos contadores están previstos para cinco cifras y, por tanto, pueden contar hasta 99.999 kilómetros; el kilómetro 100.000 hace desbordar el contador, que vuelve a indicar 000000.

La situacón es análoga en el caso de contadores electrónicos como son los campos numéricos descritos en un programa Cobol.

Para garantizar siempre la posibilidad de alcanzar el valor contenido en un contador, el Cobol prevé, para todos los verbos aritméticos, la cláusula ON SIZE ERROR. Mediante esta cláusula, el programa puede indicar el desbordamiento de un contador en el momento en que se incrementa su contenido. Tratándose evidentemente de una condición anómala que puede desvirtuar los resultados, en este caso se asocia una frase imperativa que permite al programador gestionar la situación según las exigencias del caso. En el ejemplo examinado, el desbordamiento del contador SUBTOTAL no permite gestionar de forma alternativa la situación y, por tanto, es necesario indicar a la consola que se produce este inconveniente y cerrar a continuación el proceso. De la observación del ejemplo indicado también pueden extraerse reglas generales válidas no sólo para la instrucción COMPUTE, sino para todos los demás verbos aritméticos que se analizarán a continuación.

1 / Todos los nombres usados en las expresiones numéricas deben representar datos numéricos definidos en la DATA DIVISION

2 / Todas las constantes usadas en las instrucciones aritméticas deben ser numéricas

3 / La máxima longitud prevista para cada operando de una expresión aritmética es de 18 caracteres

4 / La PICTURE y la cláusula USAGE de los diversos operandos presentes en una expresión aritmética no deben necesariamente ser uniformes, puesto que durante las operaciones se efectúan automáticamente tanto el alineado al punto decimal (V) como la conversión del formato de cada operando.

Esta conversión, como ya se ha indicado, procede según un preciso orden jerárquico (ver pág. 1014).

Considérense, por ejemplo, los campos

```
01 CAMPO-1     PIC S9(8)V9(37).
01 CAMPO-2     COMP-1.
01 CAMPO-3     COMP-2.
```

Las operaciones de conversión activadas por la instrucción

```
COMPUTE  CAMPO-3 =
         CAMPO-1 * CAMPO-2
```

son las siguientes:

- conversión de CAMPO-1 de DISPLAY a COMP-1 (efectivamente COMP-1 es la representación a la prioridad más alta entre las de los operandos presentes)
- cálculo de CAMPO-1 * CAMPO-2 en código COMP-1
- conversión del resultado anterior de COMP-1 en COMP-2.

5 / El formato de los datos utilizados en una expresión aritmética y destinados a rápidas operaciones de cálculo, no puede contener los símbolos de predisposición para la impresión.

El verbo COMPUTE puede activar las cinco operaciones fundamentales de la matemática en una serie de operandos reunidos en una única expresión. Los verbos analizados a continuación, en cambio, permiten la activación de un solo tipo de operación sobre los datos descritos.

El verbo ADD

El verbo ADD, como sugiere intuitivamente su nombre, permite sumar, según tres diferentes modalidades, dos o más cantidades numéricas, dejando disponible el resultado en un campo de llegada. Un primer formato es el indicado en la tabla de abajo.
Más allá de la aparente complejidad de la representación general, este formato es el más cercano a la estructura clásica de la suma. Efectivamente, considérense los siguientes campos

```
01 SUMANDO-1 PIC S9(4)V9(2) COMP.
01 SUMANDO-2 PIC S9(4)V9(2) COMP.
01 SUMANDO-3 PIC S9(4)V9(2) COMP.
```

Se quiere sumar su contenido y una constante física, igual a 3000, al contenido de un campo suma definido por

```
01 SUMA      PIC S9(4) COMP.
```

La operación buscada es codificable como sigue:

```
ADD  SUMANDO-1
     SUMANDO-2
     SUMANDO-3
     3000        TO SUMA.
```

Observando las descripciones de los diversos sumandos puede verse que cada uno de ellos prevé dos cifras decimales que se truncan durante la última operación de suma en el campo SUMA, ya que éste no prevé cifras decimales. Por tanto, si se desea tener en el campo SUMA un resultado redondeado y no truncado, basta con codificar de la siguiente manera:

```
ADD  SUMANDO-1
     SUMANDO-2
     SUMANDO-3
     3000        TO SUMA ROUNDED.
```

Por otra parte, el campo SUMA prevé el mismo número de caracteres de los sumandos, por lo que es muy probable que se verifique un desbordamiento en su contenido, con la consiguiente pérdida de obtención del dato. Con la misma lógica descrita en la instrucción COMPUTE, puede controlarse el eventual desbordamiento de SUMA especificando la cláusula ON SIZE ERROR e indicando, mediante una frase imperativa, el tipo de acción a emprender en el caso de que se verifique esto. Por ejemplo:

```
ADD SUMANDO-1
    SUMANDO-2
    SUMANDO-3
    3000        TO SUMA ROUNDED
                ON SIZE ERROR
                DISPLAY 'CAMPO SUMA
                         EN OVERFLOW'
                         UPON CONSOLE
                GO TO FIN.
```

PRIMER FORMATO DE LA INSTRUCCION ADD

<u>ADD</u> { nombre-de-dato-1 | constante-1 } [{ nombre-de-dato-2 | constante-2 }] ....

<u>TO</u> nombre-de-dato-n [<u>ROUNDED</u>] nombre-de-dato-p [<u>ROUNDED</u>] ....
[<u>ON SIZE ERROR</u> frase imperativa].

Debe precisarse que el campo de llegada (en este caso SUMA), como está sujeto a cálculo, debe ser un campo numérico que en la PICTURE no contenga símbolos de predisposición para la impresión. Esta última limitación se invalida adoptando el segundo formato de la instrucción ADD indicado en la primera de las dos tablas de abajo. En este caso, efectuada la suma de los diversos sumandos, el resultado se desplaza en el campo de llegada nombre-de-dato-n. Por tanto, este último, al no estar involucrado en operaciones de cálculo sino sólo de desplazamiento, puede contener símbolos de predisposición para la impresión.
El último formato de la ADD aprovecha la posibilidad del Cobol de escribir los subcampos de los campos diferentes con nombres iguales. Considérense las dos descripciones siguientes:

```
01 BLOQUE-1.
   05 MIEMBRO-1.
      10 MIEMBRO-11   PIC S9(3) COMP.
      10 MIEMBRO-12   PIC S9(2) COMP.
   05 MIEMBRO-2       PIC S9(4) COMP.
   05 MIEMBRO-3       PIC S9(6) COMP.

01 BLOQUE-2.
   05 MIEMBRO-1.
      10 MIEMBRO-11   PIC S9(3) COMP.
      10 MIEMBRO-12   PIC S9(2) COMP.
   05 MIEMBRO-2       PIC S9(4) COMP.
   05 MIEMBRO-3       PIC S9(6) COMP.
```

Si se desea sumar miembro a miembro todos los subcampos de BLOQUE-1 a los subcampos de BLOQUE-2 utilizando el primer formato de la instrucción ADD, debe escribirse:

```
ADD MIEMBRO-11     OF BLOQUE-1
TO MIEMBRO-11      OF BLOQUE-2.
ADD MIEMBRO-12     OF BLOQUE–1
TO MIEMBRO-12      OF BLOQUE-2.
ADD MIEMBRO-2      OF BLOQUE–1
TO MIEMBRO-2       OF BLOQUE-2.
ADD MIEMBRO-3      OF BLOQUE-1
TO MIEMBRO-3       OF BLOQUE-2.
```

En cambio, adoptando el tercer formato de la ADD (ver la tabla al pie de página) puede obtenerse el mismo resultado escribiendo

```
ADD CORRESPONDING BLOQUE-1
TO BLOQUE-2.
```

A pesar de la inconfundible utilidad en este caso muy particular, debe observarse que la definición de campos con nombres de subcampos iguales es desaconsejable.
Siempre en función de la legibilidad y manejabilidad del programa, es interesante asignar nombres diferentes a los campos. De este modo se puede intuir fácilmente la función lógica (del nombre) y pueden seguirse a lo largo de todo el flujo del proceso (de la impresión que el Compilador emite: CROSS REFERENCE).

SEGUNDO FORMATO DE LA INSTRUCCION ADD

ADD { nombre-de-dato-1 | constante-1 } [{ nombre-de-dato-2 | constante-2 }] ....

GIVING nombre-de-dato-n [ROUNDED] [ON SIZE ERROR frase-imperativa].

TERCER FORMATO DE LA INSTRUCCION ADD

ADD { CORRESPONDING | CORR } nombre-de-dato-1 TO nombre-de-dato-2

[ROUNDED] [ON SIZE ERROR frase-imperativa].

El verbo SUBTRACT

El verbo SUBTRACT mantiene la misma implantación lógica que el verbo ADD, y permite restar del (FROM) contenido de un determinado campo el de uno o más campos.
Todas las cláusulas que aparecen en los tres formatos de que dispone la instrucción respetan las reglas ya descritas para el verbo ADD. Con el fin de poner un ejemplo del uso del primer formato de la instrucción SUBTRACT (ver la primera tabla de abajo), considérense los campos

```
01 VALOR-INICIAL-1        PIC S9(4) COMP
   VALUE 3500.
01 VALOR-INICIAL-2        PIC S9(4) COMP
   VALUE 1000.
01 VALOR-INICIAL-3        PIC S9(4) COMP
   VALUE 0.
01 SUSTRAENDO-1           PIC S9(3) COMP
   VALUE 630.
01 SUSTRAENDO-2           PIC S9(3) COMP
   VALUE 280.
01 SUSTRAENDO-3           PIC S9(3) COMP
   VALUE 2.
```

La instrucción

```
SUBTRACT 180
         SUSTRAENDO-1
         SUSTRAENDO-2
         SUSTRAENDO-3

   FROM     VALOR-INICIAL-1
            VALOR-INICIAL-2.
```

efectúa la suma de 180, SUSTRAENDO-1, SUSTRAENDO-2, SUSTRAENDO-3 y resta el resultado del contenido de los campos VALOR-INICIAL-1 y VALOR-INICIAL-2.
En base a los valores implantados en los diversos campos, el resultado de la operación será:

```
VALOR-INICIAL-1 = 2408
VALOR-INICIAL-2 = – 92
```

Si se desea dejar sin alteración el contenido del campo en que se han efectuado las restas y obtener el resultado de la operación en otro campo, debe adoptarse el segundo formato de la instrucción (segunda tabla de abajo).

PRIMER FORMATO DE LA INSTRUCCION SUBTRACT

```
SUBTRACT  { nombre-de-dato-1 }  [ { nombre-de-dato-2 } ]  ..........
          { constante-1      }  [ { constante-2      } ]

    FROM nombre-de-dato-m [ROUNDED]
         nombre-de-dato-n [ROUNDED] ..........

         [ON SIZE ERROR frase-imperativa]
```

SEGUNDO FORMATO DE LA INSTRUCCION SUBTRACT

```
SUBTRACT  { nombre-de-dato-1 }  [ { nombre-de-dato-2 } ]  ..........
          { constante-1      }  [ { constante-2      } ]
    FROM      { nombre-de-dato-m }
              { constante-m      }
    GIVING    nombre-de-dato-n [ROUNDED]

              [ON SIZE ERROR frase-imperativa].
```

Utilizando los mismos campos del ejemplo anterior puede escribirse:

```
SUBTRACT 180
          SUSTRAENDO-1
          SUSTRAENDO-2
          SUSTRAENDO-3

      FROM      VALOR-INICIAL-1
      GIVING    VALOR-INICIAL-3.
```

El resultado de la operación será

VALOR-INICIAL-1 = 3500
VALOR-INICIAL-3 = 2048

El tercer formato (ver tabla de abajo) es completamente equivalente al análogo para el verbo ADD. Haciendo referencia a los grupos de datos BLOQUE-1 y BLOQUE-2, ya descritos para el verbo de suma, la instrucción

```
SUBTRACT CORR BLOQUE-1
FROM BLOQUE-2
```

es equivalente a las siguientes:

```
SUBTRACT  MIEMBRO-11 OF BLOQUE-1
FROM      MIEMBRO-11 OF BLOQUE-2.

SUBTRACT  MIEMBRO-12 OF BLOQUE-1
FROM      MIEMBRO-12 OF BLOQUE-2.

SUBTRACT  MIEMBRO-2 OF BLOQUE-1
FROM      MIEMBRO-2 OF BLOQUE-2

SUBRACT   MIEMBRO-3 OF BLOQUE-1
FROM      MIEMBRO-3 OF BLOQUE-2.
```

Suponiendo que los valores contenidos antes de la ejecución de la instrucción en los diversos subcampos fuesen los siguientes:

	BLOQUE-1	**BLOQUE-2**
MIEMBRO-11	210	500
MIEMBRO-12	31	63
MIEMBRO-2	4780	1200
MIEMBRO-3	370	25612

la situación después de la ejecución de la instrucción será la siguiente:

	BLOQUE-1	**BLOQUE-2**
MIEMBRO-11	210	290
MIEMBRO-12	31	32
MIEMBRO-2	4780	–3580
MIEMBRO-3	370	25242

El verbo MULTIPLY

El formato más usado es el indicado en la primera tabla de la página siguiente, en la que el resultado de la multiplicación entre el primer factor y el segundo se memoriza en el campo nombre-de-dato-3.

El verbo DIVIDE

El primer formato del verbo DIVIDE está descrito en la segunda tabla de la página siguiente. Este actúa de forma análoga al verbo MULTIPLY, o bien memoriza en un campo de llegada (nombre-de-dato-3) el resultado de la división entre el dividendo (nombre-de-dato-1 o constante-1) y el divisor (nombre-de-dato-2 o constante-2). Debe observarse que el cociente se calcula en una sección adecuada de la CPU con un número fijo de cifras decimales. Por tanto, si se necesita el resultado redondeado, la operación de redondeo del cociente se realiza después de que éste se haya calculado con el número fijo de cifras decimales previstas. También debe tenerse presente que la división de un número por cero dará lugar a un error de desbordamiento, que puede ser gestionado por el programador insertando la cláusula ON SIZE-ERROR.

TERCER FORMATO DE LA INSTRUCCION SUBTRACT

```
            ⎧ CORRESPONDING ⎫
SUBTRACT    ⎨               ⎬  nombre-de-dato-1 FROM nombre-de-dato-2
            ⎩ CORR          ⎭

[ROUNDED] [ON SIZE ERROR frase-imperativa].
```

FORMATO DE LA INSTRUCCION MULTIPLY

MULTIPLY { nombre-de-dato-1 / constante-1 } BY { nombre-de-dato-2 / constante-2 }

GIVING nombre-de-dato-3 [ROUNDED]

[ON SIZE ERROR frase-imperativa].

PRIMER FORMATO DE LA INSTRUCCION DIVIDE

DIVIDE { nombre-de-dato-1 / constante-1 } BY { nombre-de-dato-2 / constante-2 }

GIVING nombre-de-dato-3 [ROUNDED]

[ON SIZE ERROR frase-imperativa].

Es superfluo indicar que el campo que albergará el resultado deberá ser descrito en forma adecuada. Si el resultado de la operación es un número decimal y el campo de llegada está definido como entero, se tendrá la pérdida de todas las cifras decimales. En otras palabras

DIVIDE 10 BY 4 GIVING COCIENTE

tendrá como resultado

COCIENTE = 2.5

si el campo de llegada está descrito por

01 COCIENTE PIC S9V9.

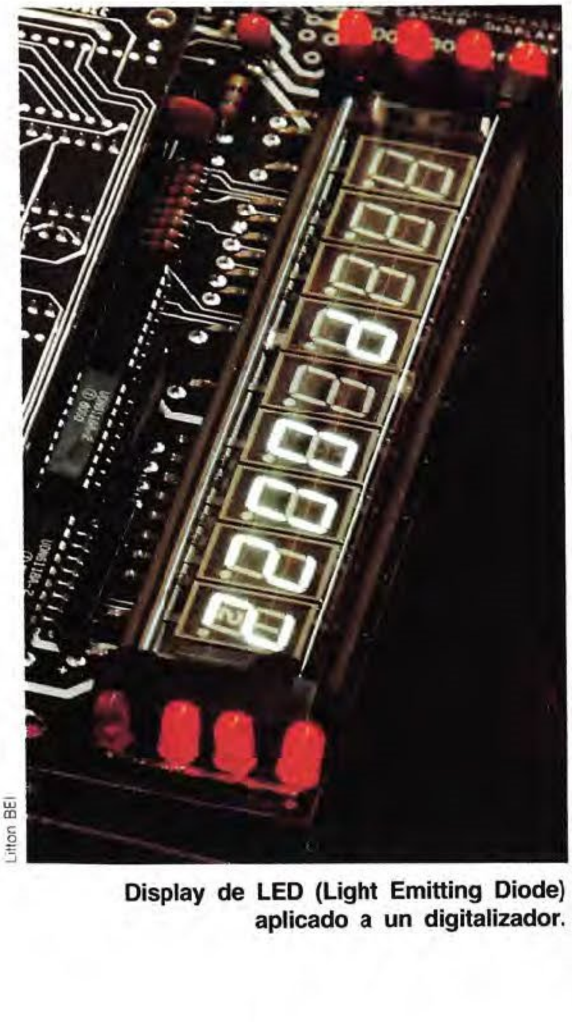
pero producirá

COCIENTE = 2

en el caso en el que COCIENTE sea descrito como sigue:

01 COCIENTE PIC S9.

En algunas aplicaciones puede ser útil tener como resultado de una división tanto la parte entera del cociente como el resto.
A esta exigencia responde el segundo formato de la DIVIDE.



Litton BEI

Display de LED (Light Emitting Diode) aplicado a un digitalizador.

SEGUNDO FORMATO DE LA INSTRUCCION DIVIDE

```
DIVIDE   { nombre-de-dato-1 }   BY   { nombre-de-dato-2 }
         { constante-1      }        { constante-2      }
         GIVING nombre-de-dato-3 [ROUNDED]
         REMINDER nombre-de-dato-4
                    [ON SIZE ERROR frase-imperativa]
```

A título de ejemplo considérese el caso en que se quiere saber si el número contenido en el campo NUMERO es par o impar.
Sabido es que todos los números pares son divisibles por dos con resto igual a 0, por lo que lo indicado puede realizarse como sigue:

```
DIVIDE NUMERO BY 2
              GIVING COCIENTE
              REMAINDER RESTO.

IF RESTO = 0
    DISPLAY 'NUMERO PAR'
            UPON CONSOLE
ELSE
    DISPLAY 'NUMERO IMPAR'
            UPON CONSOLE.
```

Verbos de transferencia y de manipulación de los datos

Una de las instrucciones más usadas y al mismo tiempo más importantes del Cobol es la instrucción MOVE, que efectúa la transferencia de datos de un campo a otro entre los escritos en la DATA DIVISION del programa. Considerado el significado autoexplicativo del verbo MOVE (desplaza), esta instrucción ya se ha usado, en su formato más sencillo, en algunos de los ejemplos indicados anteriormente.
Antes de analizar en detalle la instrucción, debe subrayarse que el Compilador determina el tipo de MOVE a efectuar en los campos interesados en base a las características del campo de llegada. Es decir, según que el campo de llegada sea alfanumérico o numérico, la MOVE será alfanumérica o numérica y, por tanto, trabajará según modalidades precisas de transferencia, diferentes entre tipo y tipo.
También existe un tercer tipo de MOVE, llamado **MOVE por prospecto**, orientado a la transferencia de datos en campos cuya PICTURE contiene caracteres de predisposición para la impresión **(símbolos de editing)**. Estos símbolos permiten insertar o enmascarar algunos caracteres del contenido del campo en función de las exigencias estéticas, funcionales o de seguridad según como se haya estructurado la impresión.
Considérese por ejemplo el caso en que el campo TOTAL descrito por

```
01 TOTAL          PIC 9(5)V9(4).
```

deba indicarse en una fila de impresión. Porque, como se sabe, el caracter V representa sólo una posición virtual del punto decimal; si el campo en impresión tuviese la misma PICTURE no se sabría como establecer el valor real de TOTAL. Efectivamente, si el valor fuese 45781.2478, en impresión se tendría

457812478

Transfiriendo TOTAL en el campo TOTAL-IMPRIME

```
01 TOTAL-IMPRIME    PIC 9(5).9(4).
```

en cambio se tiene la representación real del número contenido en TOTAL. Por tanto, el carácter · (punto) es un carácter de predisposición a la impresión. Observemos que contrariamente al símbolo V, el punto ocupa una posición de memoria: la ocupación total de TOTAL-IMPRIME es de 10 caracteres.

El verbo MOVE

La instrucción MOVE tiene dos formatos (ver las tablas de la página siguiente).
Es interesante subrayar que los dos formatos

PRIMER FORMATO DE LA INSTRUCCION MOVE

```
        ⎧ nombre-de-dato-1 ⎫
  MOVE  ⎨                  ⎬  TO   nombre-de-dato-2 [nombre-de-dato-3].........
        ⎩ constante        ⎭
```

SEGUNDO FORMATO DE LA INSTRUCCION MOVE

```
        ⎧ CORRESPONDING ⎫
  MOVE  ⎨               ⎬   nombre-de-dato-1 TO nombre-de-dato-2.
        ⎩ CORR          ⎭
```

son válidos tanto para MOVE alfanuméricas como para MOVE numéricas o por prospecto, y que la calificación de la instrucción está ligada a las descripciones de los campos de llegada interesados.

En general, la MOVE transfiere el contenido de nombre-de-dato-1 o una constante en uno o más campos de llegada.

Considérese por ejemplo el campo

```
01 PARTIDA         PIC X(20).
```

y se quiere transferir el contenido en:

```
01 LLEGADA-1       PIC X(20).
01 LLEGADA-2       PIC X(20).
01 LLEGADA-3       PIC X(20).
```

En este caso puede escribirse

```
MOVE PARTIDA TO LLEGADA-1.
MOVE PARTIDA TO LLEGADA-2.
MOVE PARTIDA TO LLEGADA-3.
```

o bien

```
MOVE PARTIDA TO LLEGADA-1
                LLEGADA-2
                LLEGADA-3.
```

Las dos escrituras son completamente equivalentes, puesto que en la segunda, el Compilador desarrolla la forma abreviada en tres MOVE separados.

Por tanto se aconseja adoptar la forma abreviada, puesto que se hace más evidente la operación de MOVE simultánea del mismo campo a más campos de llegada. En el formato

```
MOVE CONSTANTE TO nombre-de-dato-2
                  nombre-de-dato-3
                  ..................
```

«constante» debe ser una constante alfanumérica o numérica según cuál sea el tipo de campo de llegada.

En base a lo dicho a propósito de las MOVE múltiples, es evidente que la compatibilidad debe ser respetada para cada uno de los campos de llegada. El siguiente ejemplo

```
WORKING-STORAGE SECTION.
01 LLEGADA-1                 PIC X(20).
01 LLEGADA-2.
   05 CAMPO-1                PIC 9(10).
   05 FILLER                 PIC X(10).
PROCEDURE DIVISION.
INICIO.
MOVE
'INICIO PROCESO' TO LLEGADA-1.
                    CAMPO-1.
```

es evidentemente erróneo, puesto que la constante alfanumérica INICIO PROCESO no puede ponerse en un campo numérico como CAMPO-1, mientras que es correcto moverla en LLEGADA-1.

La transferencia de los datos en uno o más campos de llegada no destruye el contenido del campo de partida, que queda inalterado hasta que no se modifique voluntariamente. Tanto el campo de partida como el de llegada pueden ser campos elementales o compuestos.

Recordemos a propósito que un dato compuesto es asumido en su conjunto como alfanumérico, independientemente de las descripciones de los datos componentes.

Seguidamente se analiza el siguiente ejemplo:

```
DATA DIVISION.
FILE SECTION.
FD FICHA
   LABEL RECORD OMITTED.
01 FICHA                        PIC X(3).
   05 TIPO-FICHA                PIC X(3).
   05 DESCRIPCION.
      10 NOMBRE                 PIC X(10).
      10 APELLIDO               PIC X(15).
   05 FECHA-NACIMIENTO.
      10 DIA                    PIC 9(2).
      10 MES                    PIC 9(2).
      10 AÑO                    PIC 9(2).
   05 SEXO                      PIC X.
   05 FILLER                    PIC X(45).
*
*
 WORKING-STORAGE SECTION.
77 AUXILIAR-FICHA               PIC X(80).
78 AUXILIAR-DESCRIPCION         PIC X(25).
01 ....................
   ....................
*
*
 PROCEDURE DIVISION.
 INICIO.
    ....................
    ....................
*
*** EJEMPLOS DE TRANSFERENCIA
*
*
MOVE SPACES        TO AUXILIAR-FICHA
MOVE FICHA         TO AUXILIAR-FICHA
MOVE AUXILIAR-DESCRIPCION
                   TO DESCRIPCION.
MOVE 'ABC'         TO TIPO-FICHA.
MOVE 83            TO AÑO.
...............
...............
```

Como puede observarse:

```
MOVE SPACES TO AUXILIAR-FICHA
```

desplaza la constante figurativa SPACES en un campo elemental

```
MOVE FICHA TO AUXILIAR-FICHA
```

transfiere el campo compuesto FICHA en un campo elemental

```
MOVE AUXILIAR-DESCRIPCION
     TO DESCRIPCION
```

desplaza un campo elemental en un campo compuesto

```
MOVE 'ABC' TO TIPO-FICHA
```

transfiere la constante alfanumérica ABC en un campo elemental

```
MOVE 83 TO AÑO
```

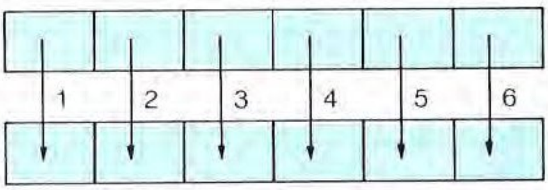
implanta en el campo AÑO la constante numérica 83.

Para entrar en el análisis detallado de la instrucción MOVE es necesario presentar una importante precisión correspondiente a las constantes figurativas. Como ya se ha indicado, éstas son:

```
SPACE        (SPACES)
ZERO         (ZEROS o ZEROES)
LOW-VALUE    (LOW-VALUES)
HIGH-VALUE   (HIGH-VALUES)
QUOTE        (QUOTES)
ALL
```

De estas, SPACE (o SPACES) se considera alfabética, ZERO (o ZEROS, o ZEROES) se asimila a una constante numérica, mientras que LOW-VALUE y HIGH-VALUE se consideran como pertenecientes a la categoría de los datos alfanuméricos.

MOVE alfanumérica. La MOVE alfanumérica efectúa la transferencia de los datos en la secuencia indicada en el siguiente esquema



o bien, el campo de llegada se llena de izquierda a derecha copiando carácter por carácter el contenido del campo de partida.

En el caso en que el campo de partida tenga una longitud inferior a la del campo de llegada, este último se llena con blanks.



Perkin-Elmere

Empleo del procesador electrónico en una bolsa de valores.

Por ejemplo,

```
01 PARTIDA     PIC X(3) VALUE 'ABC'.
01 LLEGADA     PIC X(6) VALUE 'UVZXYW'.
PROCEDURE DIVISION.
A.
   MOVE PARTIDA TO LLEGADA.
```

La situación antes de la instrucción MOVE es la siguiente

PARTIDA | A | B | C |

LLEGADA | U | V | Z | X | Y | W |

mientras que inmediatamente después es

PARTIDA | A | B | C |

LLEGADA | A | B | C | | | |

Como puede observarse, la transferencia de los caracteres de PARTIDA en LLEGADA ha provocado el llenado del campo con blanks a la derecha que han recubierto el contenido anterior.
Si se desea invertir esta modalidad de transferencia (o bien hacer de manera que el campo LLEGADA sea llenado con el contenido de PARTIDA empezando desde la derecha y llenado con blanks a la izquierda) el campo de llegada debe contener en la propia descripción la cláusula JUSTIFIED RIGHT (JUST).
Utilizando todavía el campo del ejemplo anterior se tiene:

```
01 PARTIDA     PIC X(3) VALUE 'ABC'.
01 LLEGADA     PIC X(6) JUST VALUE
               'UVZXYW'
PROCEDURE DIVISION.
A.
       MOVE PARTIDA TO LLEGADA.
```

Situación antes de la MOVE

PARTIDA | A | B | C |

LLEGADA | U | V | Z | X | Y | W |

después de la MOVE

PARTIDA | A | B | C |

LLEGADA | | | | A | B | C |

Considérese ahora el caso en que el campo de partida tenga una longitud superior a la del campo de llegada. Por ejemplo,

```
01 PARTIDA    PIC X(6) VALUE 'ABCDEF'.
01 LLEGADA    PIC X(3) VALUE SPACES.
```

En este caso, la situación que precede a la ejecución de la instrucción MOVE PARTIDA TO LLEGADA es la siguiente

PARTIDA | A | B | C | D | E | F |

LLEGADA | | | |

mientras que, después de la ejecución de la instrucción, se tiene

PARTIDA | A | B | C | D | E | F |

LLEGADA | A | B | C |

o sea se ha producido un truncado a la derecha de todos los caracteres que excedían las dimensiones de LLEGADA.

En cambio, si la definición del campo LLEGADA hubiese sido

```
01 LLEGADA PIC X(3) JUST VALUE SPACES.
```

la situación final después de la instrucción MOVE habría sido

PARTIDA | A | B | C | D | E | F |

LLEGADA | D | E | F |

Es decir, el truncado de los caracteres en exceso se habría producido a la izquierda, después del llenado (a la derecha) del espacio disponible. Una síntesis de la MOVE de tipo alfanumérico descrita hasta ahora puede verse en la tabla de abajo.
De la observación de la tabla puede verse que un campo compuesto puede albergar datos procedentes de cualquier otro tipo de campo.
Una excepción a esta regla general viene de los números en coma flotante (USAGE COMP-1 y COMP-2).
Esta excepción no está evidenciada en la tabla; sin embargo, es interesante subrayar el caso con el siguiente ejemplo.
Considérense los dos campos

```
01 NUMERO              COMP-1.
01 CAMPO.
   05 SUBCAMPO-1       PIC 9(4).
   05 SUBCAMPO-2       PIC X(3).
   05 FILLER           PIC X.
```

CAMPO DE PARTIDA — **CAMPO DE LLEGADA**

Tipo	Compuesto	Elemental Alfanumérico	Elemental Alfabético	Elemental Numérico
Compuesto	SI	SI	SI	NO
Elemental Alfanumérico	SI	SI	SI	NO
Elemental Alfabético	SI	SI	SI	NO
Elemental Numérico	SI	SI*	NO	SI

* Sólo si el campo de partida es entero

La instrucción

MOVE NUMERO TO CAMPO

es errónea, puesto que el campo NUMERO tiene la codificación interna en coma flotante de simple precisión.
Además obsérvese que el asterisco que aparece en la tabla es para indicar que la transferencia de un campo elemental numérico en un campo alfanumérico sólo es posible si el campo de partida es entero.
La USAGE de este campo puede ser indiferentemente COMP o DISPLAY; también en este caso permanece la imposibilidad de utilizar como campo de partida un campo de coma flotante.
Durante la transferencia de un campo numérico a un campo alfanumérico se efectúa la conversión en el formato del campo de llegada, de acuerdo con las modalidades descritas para este tipo de campos. Para aclarar esto, considérese el campo

```
01 LLEGADA                PIC X(3).
```

y las instrucciones

```
MOVE 38 TO LLEGADA.
MOVE 7425 TO LLEGADA.
```

Después de la primera MOVE, el contenido de llegada será

3	8	

mientras que después de la segunda, el campo tendrá la siguiente configuración

7	4	2

Debe observarse explícitamente que en el caso en que se desee controlar si el campo LLEGADA contiene el número 38, la verificación debe hacerse sobre los tres caracteres del campo. Es decir, la secuencia de instrucciones

```
MOVE 38 TO LLEGADA.
IF LLEGADA = '38'
   PERFORM IGUAL-38.
```

nunca conducirá a la ejecución de la SECTION IGUAL-38.



J. Pickerell/Marka

Prueba de las placas de unidades para disco flexible.

Efectivamente, después de la ejecución de la instrucción MOVE 38 TO LLEGADA, el contenido el campo LLEGADA tendrá la siguiente configuración:

3	8	ƀ

y no 38; la forma correcta del ejemplo anterior es la siguiente:

```
MOVE 38 TO LLEGADA.
IF LLEGADA = '38 '
   PERFORM IGUAL-38.
```

Debe observarse explícitamente que lo expuesto es válido también si el campo LLEGADA está justificado a la derecha (JUSTRIGHT); es decir, ' 38' no es igual a '38'.

La cláusula ALL. Si se tiene necesidad de llenar todo un campo con una serie de caracteres, existe el siguiente formato de la MOVE:

<u>MOVE</u> <u>ALL</u> caracteres <u>TO</u> nombre-de-campo.

El diagnóstico computerizado (3)

La ecografía es un método de diagnóstico basado en la producción de imágenes de los detalles anatómicos mediante sondas especiales que emiten ondas ultrasonoras. Los ecos reflejados en diferente medida por las diversas estructuras anatómicas se procesan con un microprocesador y se proyectan en una pantalla, sobre la que se crea una imagen que reproduce fielmente las características morfológicas y estructurales de la parte o del órgano examinado. Las imágenes que aparecen en la pantalla pueden fotografiarse y analizarse cómodamente, y permiten formular juicios y emitir diagnósticos con gran precisión y fiabilidad.

Nació a principios de los años setenta, pero sólo en la mitad de la última década fueron desarrolladas en Europa y en EE.UU. las técnicas que permitieron añadir a los actuales aparatos el proceso en tiempo real de imágenes de órganos en movimiento. Las otras ventajas de los aparatos ecográficos son el coste limitado con respecto a otros sistemas de exploración, las reducidas dimensiones y la facilidad de empleo.

Actualmente, las unidades ecográficas multidisciplinarias como la SDU 3000 están dotadas de una amplia gama de transductores para exámenes estáticos y dinámicos de tipo pediátrico, oftalmológico, obstétrico y para el examen de órganos profundos y superficiales en general. Existen fundamentalmente dos tipos de sondas para exploraciones dinámicas: el tipo lineal de doble focalización (geométrica y electrónica), para exámenes panorámicos del abdomen y para obstetricia, y el tipo de sector, que permite eliminar las dificultades de acceso a órganos y aparatos determinados. Las variaciones sobre el tema, por tanto, son innumerables.

Los sistemas multidisciplinarios de base están dotados de videocinta incorporado, de fotocámaras para la producción de copias sobre papel, de sondas biópticas y de perforadoras para el registro de los datos en cinta de papel, todo contenido en un pequeño mueble del tamaño de un frigorífico, cómodamente transportable. Normalmente se alimentan con la tensión de red, y tienen unas características de resolución y penetración que parecían una meta utópica para los imponentes aparatos que se construían hace menos de diez años. Una sonda sectorial tiene una penetración que puede alcanzar 30 cm y una resolución angular de sólo 12'. Pero los límites de la miniaturización todavía están bien lejos de haberse alcanzado. El sistema Sono Diagnost R1000 es la demostración palpable: basta pensar que pesa menos de 12 kg, comprendida la fotocámara (ver foto de abajo). El monitor presenta imágenes de 10 × 8 cm producidas procesando la exploración tomada 25 veces por segundo de una sonda lineal que emite un haz ultrasonoro de 2,2 a 3 MHz. Si se desea, el aparato también puede funcionar en el campo, puesto que basta conectarlo a la batería del automóvil. El precio y la sencillez de operación ponen los sistemas de este tipo al alcance de todos los consultorios médicos.

Las técnicas de exploración de rayos X y las que utilizan ultrasonidos dan una imagen morfológica de los órganos examinados que se limita a la reproducción, si bien minuciosa, del detalle anatómico, aunque no proporciona informaciones sobre su funcionalidad. El diagnóstico nuclear nació precisamente para proporcionar una respuesta a esta última exigencia, y tiene como

El Sono Diagnost R1000, aparato portátil para exploraciones ecográficas.



Philips



Philips



Philips

Aparato ecográfico transportable Sono Diagnost Universal 3000 (SDU 3000).

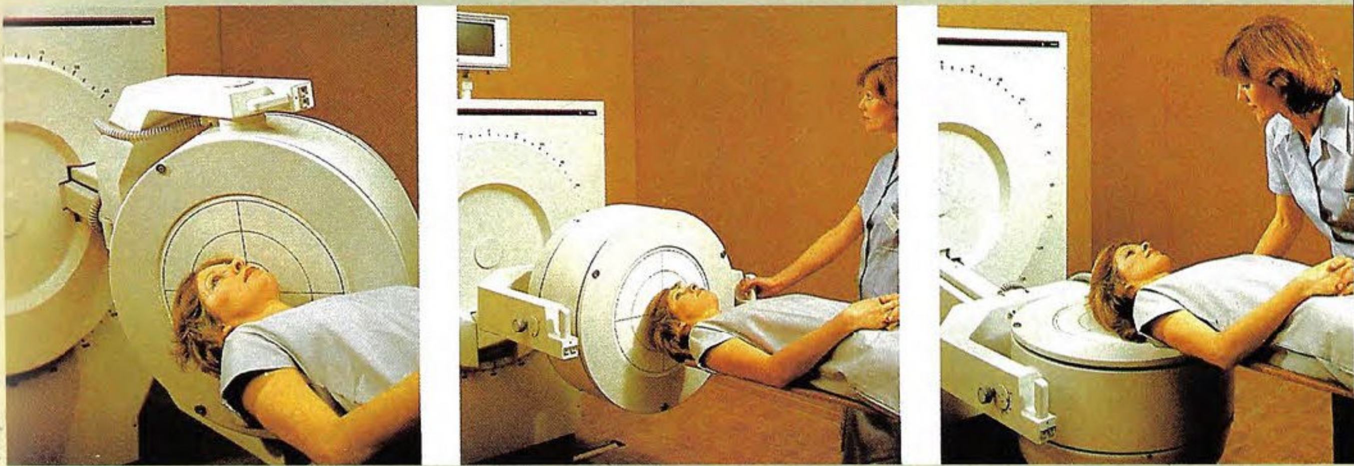
finalidad principal el estudio de la funcionalidad fisiológica de los órganos, sin limitarse a la producción de una imagen anatómica estática. La exploración de tipo nuclear requiere la introducción preliminar en el organismo de trazadores radiactivos gamma que se metabolizan en el órgano o en el particular sistema anatómico objeto de exploración. Seguidamente se procede a la detección y a la medición de la intensidad de las radiaciones emanadas por los trazadores, con lo cual es posible seguir todo el proceso de metabolización, con la síntesis de un mapa de las concentraciones del trazador en los distintos órganos o en las diversas partes de un mismo órgano.

El diagnóstico fundado en el empleo de los radioisótopos induce dosis de radiación inferiores a las que se aplican a los pacientes en el curso de una radiografía normal. También por este motivo el screening *de tipo nuclear tiende a difundirse cada vez más, especialmente en aplicaciones que contemplan el estudio del desarrollo de los procesos fisiológicos que se produce en los órganos.*

Las novedades derivan también en ese sector de las posibilidades ofrecidas por la computerización de los aparatos. Desde hace veinticinco años, el instrumento fundamental de un laboratorio de medicina nuclear es la gamma-cámara, *pero sólo en los últimos cinco años, la aplicación del procesador en el control de la exploración ha transformado este instrumento en un potente sistema de diagnóstico, con el cual es posible detectar también imágenes tomográficas axiales.*

En el sistema Gamma diagnost tomo, *que prevé la posibilidad de realizar el* total body scanning *y la tomografía axial digitalizada, el detector gamma propiamente dicho está suspendido por un brazo articulado y está dotado de un cristal destellador de un diámetro de 40 cm por 9 mm de espesor. Inmediatamente delante del cristal se encuentra una batería de 61 fotomultiplicadores, que transforma las señales luminosas, producidas en el cristal de la detección de los rayos gamma emitidos por el trazador, en señales eléctricas, las cuales a su vez se procesan para proporcionar la imagen deseada.*

El sistema de adquisición de datos tiene una velocidad de repetición de 40 imágenes por segundo, con posibilidad de adaptarse simultáneamente a dos gamma-cámaras. La consola de control y de adquisición y proceso de los da-



Philips



Philips

Arriba, posicionado del detector de la gamma-cámara «Gamma Diagnost Tomo». A la izquierda, imagen tomográfica digitalizada proporcionada por los circuitos de proceso de la gamma-cámara; se presenta una sección abdominal. En la parte derecha de la pantalla se ve la escala de concentración del trazador.

tos está provista de un monitor de 512 × 512 pixels, expandible a 640 × 256 para el total-body, *y hay disponibles 256 niveles de intensidad. El sistema de proceso incluye dos microprocesadores especializados* gamma processor 100 *en paralelo, con 400 kbytes de memoria RAM expandibles a 528, programables en Fortran, RTL2, Assembler y PMCL (un lenguaje de programación creado expresamente para las aplicaciones biomédicas). El hardware se completa con dos memorias masivas, una de discos de 24 Mbytes y otra de* floppy-disks *de 1,2 Mbytes. El procesador permite, como se ha dicho, la formación de la imagen tomográfica digitalizada, que puede presentarse en un monitor de color (foto grande de encima). La posibilidad de terminar la línea periférica de la sección examinada permite, entre otras cosas, corregir los datos teniendo en cuenta la absorción de los rayos gamma por parte de los tejidos, mejorando de forma determinante la veracidad del mapa de distribución del trazador. Imágenes tomadas en diferentes momentos pueden llamarse simultáneamente en el monitor para verificar, con una simple mirada, la evolución del proceso de concentración del trazador en un determinado órgano. Además, el aparato está completado con un sistema para la producción de copias en papel o en soporte transparente.*

Sistemas como el ilustrado han permitido eliminar los riesgos asociados a los tradicionales sistemas de verificación de la funcionalidad de los órganos, como en el caso del cateterismo cardiaco, y proporcionan una contribución de primer orden a la diagnosis de las afecciones de los pulmones, de los riñones, el corazón, el hí-



Philips

El panel de mando y de control de un aparato para la termografía de imágenes digitalizadas. A la derecha pueden verse el joystick de posicionado y los cursores de delimitación del campo de observación.

gado, el páncreas, el sistema óseo y el sistema endocrino.

La termografía *es una técnica de exploración relativamente reciente. El funcionamiento de los termógrafos de la primera generación (se habla sólo de algunos años) se basaba en el empleo de placas fotográficas sensibles al inflarrojo, que se exponían e impresionaban directamente con el calor corporal del paciente. Así se aprovechaba la propiedad común a muchos fenómenos inflamatorios y patológicos de alterar localmente la temperatura corporal, incluso si es de fracciones de grado.*

Actualmente, esta técnica de exploración se basa en el empleo de células fotovoltaicas sensibles al infrarrojo y en circuitos de proceso numérico de las señales emitidas por éstas. En lugar de las imágenes fotográficas se generan imágenes en colores en un monitor de grandes dimensiones, eventualmente archivables en copia sobre papel o en forma digitalizada en una memoria masiva.

La estructura del panel de control de un termógrafo de presentación digital se reproduce arriba. La cámara termográfica de alta resolución está equipada con una célula fotovoltaica de indio-antimonio refrigerada con nitrógeno líquido; está montada sobre un sistema de soporte que incluye los monitores eléctricos que controlan los movimientos de elevación (± 25°), de orientación (± 100°) y de nivel (50 a 180 cm). La resolución espacial de la célula permite distinguir detalles subtendidos por un ángulo inferior a 0,002 radianes (aproximadamente 7'), mientras que la resolución térmica permite diferenciar detalles cuya temperatura sólo difiere 0,08 °C. El tiempo de exposición necesario para que la célula pueda proporcionar una respuesta completa es de 1 s.

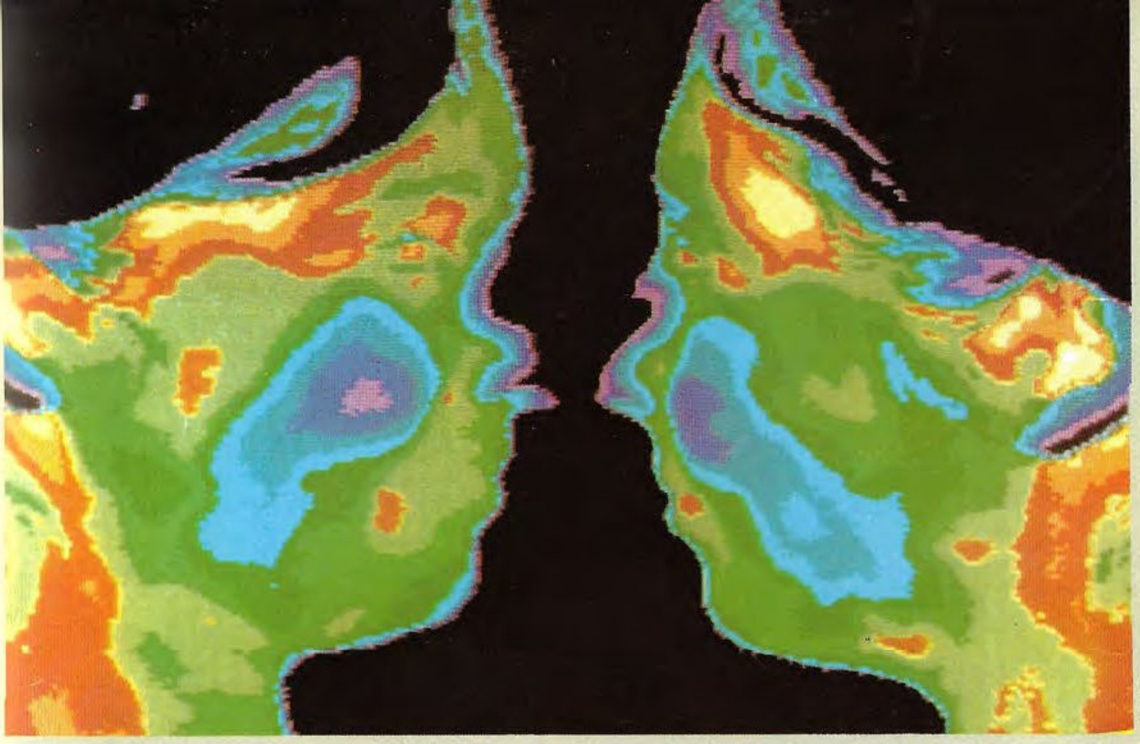
Las señales emitidas se envían a través de un cable al sistema electrónico, que procesa una matriz numérica del campo de observación y la envía a un monitor de diez colores de 12". Sobre este último se presenta una matriz de 312 × 240 pixels, junto con la escala de los colores y con indicaciones que permiten calibrar con exactitud la escala de las temperaturas.

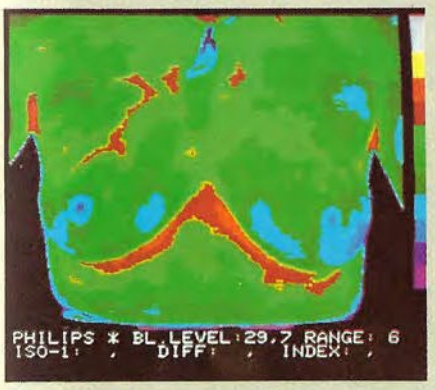
La temperatura de umbral (black level) *puede regularse con continuidad entre 15 y 40 °C; todos los detalles cuya temperatura resulta inferior al* black level *se presentan en negro. La amplitud del campo por encima del* black level (range)*, en cambio, puede regularse separadamente a los valores 2, 3, 4, 5, 8, 10, 15 °C.*

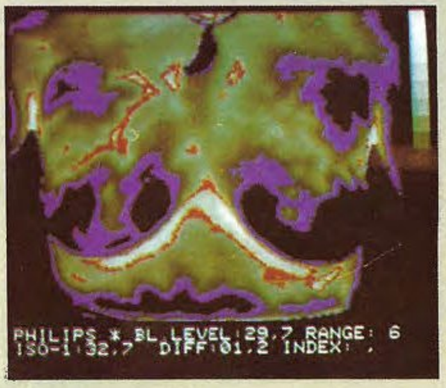
Después de presentar la imagen termográfica es posible memorizarla y reclamarla sobre una parte del monitor. Esto permite, entre otras cosas, evidenciar en el mismo momento detalles anatómicos que no pueden entrar simultáneamente en el campo visual de la cámara, como por ejemplo los dos lados de la cabeza (foto de la parte superior de la página siguiente) o el dorso y la palma de una mano.

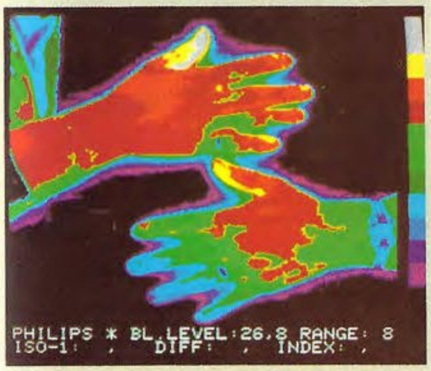
La imagen presentada puede procesarse de diferentes maneras. Una primera función del circuito electrónico permite presentar dos isotermas, cada una de amplitud igual al 3% del range *regulado, en dos colores diferentes. Elegido el nivel de la primera isoterma (ISO-1), el de la segunda puede variarse a voluntad en el* range *superior, y la diferencia de nivel entre las dos se presenta automáticamente en el monitor (DIFF).*

Unos cursores adecuados en el panel de control permiten posicionar en el monitor un recuadro sobre una región de interés particular, para la cual el calculador determina en tiempo real dicho índice termográfico *(INDEX). Pero la característica más interesante la ofrece la posibili-*











La imagen termográfica computerizada se presenta en pantalla de televisión y puede registrarse en soporte magnético. Al lado, arriba, dos imágenes termográficas diferentes de un tórax femenino; debajo, las manos de un paciente en una comparación termográfica: la reducida vascularización de la mano izquierda puede cuantificarse mediante el perfil térmico visible en la foto de al lado. Arriba se ha realizado una comparación análoga entre los dos lados del rostro.

dad de calcular y presentar el perfil térmico referido a una línea horizontal que puede situarse libremente en el monitor (foto de arriba). La curva de temperatura se procesa en base a las señales térmicas digitalizadas que se refieren a los pixels cubiertos por la línea de selección.

Las características de los aparatos termográficos digitalizados han ampliado las capacidades de diagnóstico de la termografía. Además de la tradicional aplicación al diagnóstico del cáncer de mama, tener termogramas en tiempo real, separados por un segundo, permite cuantificar también las variaciones térmicas debidas al suministro de fármacos, obteniendo datos útiles para efectuar estudios en profundidad en los campos reumatológico, fisiológico, etc.

Considérese por ejemplo el campo

01 LLEGADA PIC X(6).

Para mayor claridad indicamos en los siguientes ejemplos tanto los efectos de la MOVE ALL como los de la MOVE sencilla, al frente del mismo carácter o cadena de caracteres a transferir.

MOVE ALL '*' TO LLEGADA

*	*	*	*	*	*

MOVE '*' TO LLEGADA

*					

'MOVE ALL '6' TO LLEGADA

6	6	6	6	6	6

MOVE '6' TO LLEGADA

6					

MOVE ALL '31' TO LLEGADA

3	1	3	1	3	1

MOVE '31' TO LLEGADA

3	1				

MOVE ALL' ' TO LLEGADA

MOVE ' ' TO LLEGADA

MOVE ALL '345' TO LLEGADA

	3	4	5		3

La última instrucción muestra cómo el Compilador respeta las repeticiones de la cadena de caracteres indicada hasta el llenado del campo, truncando los caracteres en exceso.
Obsérvese que los blanks en la cadena de caracteres a transferir se tratan como cualquier otro carácter, sea cual sea la posición en que se encuentren en el ámbito de la combinación.

MOVE numérica. Como ya se ha subrayado, la operación de transferencia de datos necesita un buen conocimiento de los mecanismos que la gobiernan. Saber cómo el Compilador «instruye» el calculador en la ejecución de estas instrucciones tiene una importancia particular en la transferencia de los datos numéricos, puesto que la veracidad de los resultados está directamente ligada al correcto uso de las instrucciones y al adecuado dimensionado de los campos interesados.
Téngase presente que, mientras la MOVE de tipo alfanumérico descrita anteriormente no entra en el significado de los datos, la MOVE numérica debe tener en cuenta la USAGE del campo de llegada y otros factores necesarios para un correcto tratamiento de los datos, como el signo y la posición del punto decimal.
La transferencia de tipo numérico se realiza siguiendo cinco pasos funcionales:

1 / Alineado al punto decimal
2 / Llenado del campo de llegada
3 / Truncado de las cifras en exceso o completado del campo de llegada
4 / Conversión del formato de partida en el de llegada
5 / Tratamiento del signo.

1 / Alineado al punto decimal. Si se quiere transferir un número decimal en un campo dotado de PICTURE adecuada, es intuitivo que las cifras enteras deberán ocupar el espacio reservado para ellas a la izquierda del punto decimal y las cifras decimales a la derecha.
Supóngase que debe transferirse el campo NUM-PARTIDA en NUM-LLEGADA, donde:

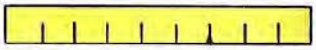
01 NUM-PARTIDA PIC S9(4)V9(3).
01 NUM-LLEGADA PIC S9(6)V9(3).

la instrucción

MOVE NUM-PARTIDA TO NUM-LLEGADA.

realiza el alineado según el esquema siguiente (el símbolo ▲ representa la posición virtual del punto decimal en la memoria):

NUM-PARTIDA 

NUM-LLEGADA 

Debe observarse que el alineado al punto decimal se realiza normalmente, incluso en el caso en que la PICTURE del campo de partida y del campo de llegada no tengan el símbolo V. Para el Compilador, las descripciones

01 NUMERO-1 PIC S9(4).
01 NUMERO-2 PIC SP(3)9(5).
01 NUMERO-3 PIC S99P(5).

son equivalentes a las que ahora se indican:

01 NUMERO-1	PIC S9(4)V.
01 NUMERO-2	PIC SVP(3)9(5).
01 NUMERO-3	PIC S99(5)V.

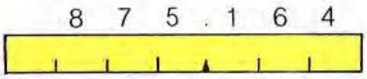
Es evidente que a los campos del ejemplo, entendidos en la forma anterior, se les pueden aplicar las modalidades de alineado anteriormente descritas.
Por otra parte, téngase en cuenta el hecho de que el punto decimal presente en una constante numérica está alineado de la misma manera que una posición virtual. Por ejemplo, si el campo LLEGADA está descrito por

01 LLEGADA PIC S9(4)V9(3).

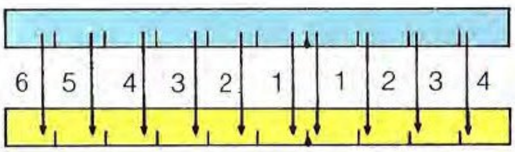
la instrucción

MOVE 875.164 TO LLEGADA.

comporta el alineado esquematizado en la siguiente figura:



2 / Llenado del campo de llegada. Después de haber alineado el punto decimal de los dos campos interesados, la parte entera del campo de llegada se llena partiendo de la derecha y procediendo hacia la izquierda, mientras que en la decimal, las cifras se copian partiendo de la izquierda.
El llenado del campo de llegada se produce según la secuencia indicada en el siguiente esquema:



3 / Truncado o completado del campo de llegada. Análogamente a lo que sucede en la MOVE de tipo alfanumérico vista anteriormente, si los campos interesados no tienen la misma longitud, se produce el llenado hasta completar el campo de llegada o el truncado de las cifras que exceden la máxima longitud del campo receptor.
Considérense los dos campos



Panel de control del calculador Univac 1100/8.

J. Pickerell/Marka

01 NUM-PARTIDA	PIC S9(3)V9 VALUE 436.2.
01 NUM-LLEGADA	PIC S9(6)V9(3) VALUE 123456.789.

La configuración de los dos campos antes y después de la ejecución de la instrucción MOVE NUM-PARTIDA TO NUM-LLEGADA es la siguiente

	NUM-PARTIDA	**NUM-LLEGADA**
antes de la MOVE	4 3 6 . 2	1 2 3 4 5 6 . 7 8 9
después de la MOVE	4 3 6 . 2	0 0 0 4 3 6 . 2 0 0

Como puede observarse, al ser el campo de

partida de longitud inferior con respecto a la del campo de llegada, se ha tenido un llenado de este último a la izquierda y a la derecha de las cifras transferidas, con ceros no significativos.
En el caso opuesto, si el campo de partida tiene una longitud superior a la del campo receptor, se realiza un truncado del valor transferido. Este truncado, teniendo presente la modalidad general de transferencia de los datos en campos numéricos, puede afectar tanto a las cifras enteras como a las decimales.
Para aclarar el concepto considérense los ejemplos indicados en la figura de abajo, que tienen por objeto los campos

```
01 NUM-PARTIDA            PIC S9(4)V9(2)
                          VALUE 1436.78.
01 LLEGADAS.
   05 NUM-LLEGADA-1       PIC S9(3)V9(2).
   05 NUM-LLEGADA-2       PIC S9(3)V9(4).
   05 NUM-LLEGADA-3       PIC S9(4).
   05 NUM-LLEGADA-4       PIC SV9.
   05 NUM-LLEGADA-5       PIC S9(6).
```

En estos ejemplos puede verse que una operación de transferencia con un campo de llegada dimensionado incorrectamente puede conducir a resultados muy alejados de los esperados.
Efectivamente, el truncado efectuado por la MOVE, si se aplica a la parte entera del número, hace perder las cifras más significativas, alterando completamente el valor tratado.
Es muy esclarecedor el último ejemplo indicado, en el que el valor 1000000.796 después de la transferencia se convierte en 0.

4 / Conversión del formato. Como ya se ha indicado, en la transferencia de los datos en campos numéricos, el Compilador puede tratar campos con codificación interna diferente.
Es decir, conociendo la USAGE implícita o explícita de los campos de partida y de llegada, el Compilador puede convertir oportunamente los datos en el formato que se desea tener en el campo receptor.
La conversión de formato se realiza, sobre el valor ya truncado eventualmente del dato, en un área interna del Compilador.
Por tanto, es intuitivo que obligar al Compilador a convertir una gran cantidad de datos equivale a penalizar la eficiencia del programa; la ejecución de las adecuadas rutinas de conversión por un lado, y la ocupación de memoria adicional dedicada a las áreas de conversión por otro, conducen a un incremento del tiempo de proceso y de la ocupación de memoria central.
Por tanto, es aconsejable describir los campos numéricos de manera homogénea, recurriendo a la conversión de formato sólo cuando no sea posible proceder de otra forma.

5 / Tratamiento del signo algebraico. La última operación que el Compilador efectúa en el campo de llegada es la correspondiente al tratamiento del signo algebraico.
Si el campo contiene en la propia PICTURE el símbolo S, el signo algebraico del valor transferido se codifica oportunamente de acuerdo con la codificación interna utilizada. Y viceversa, en ausencia del símbolo S, la instrucción MOVE sólo transfiere el valor absoluto del dato.
Se aconseja utilizar siempre el símbolo S en la descripción de campos numéricos. Esto es posible sin ninguna contraindicación, salvo poquísimas excepciones que se indicarán a continua-

TRUNCADOS EN LA TRANSFERENCIA DE DATOS

Instrucciones de transferencia	**Contenidos del campo de llegada después de la MOVE**
MOVE NUM-PARTIDA TO NUM-LLEGADA-1	4 3 6 . 7 8
MOVE NUM-PARTIDA TO NUM-LLEGADA-2	4 3 6 . 7 8 0 0
MOVE NUM-PARTIDA TO NUM-LLEGADA-3	1 4 3 6 .
MOVE NUM-PARTIDA TO NUM-LLEGADA-4	. 7
MOVE 1000000,796 TO NUM-LLEGADA-5	0 0 0 0 0 0 .

INSTRUCCION MOVE

CAMPO DE PARTIDA / CAMPO DE LLEGADA				Compuesto	Elemental									
					Alfanumérico	Alfabético	Numérico					Número editing	Alfanumérico de editing
							DISPLAY	COMP	COMP-3	COMP-1	COMP-2		
Compuesto				A	A	A						P	P
Elemental	Alfanumérico			A	A	A						P	P
Elemental	Alfabético			A	A	A							P
Elemental	Constante	Alfanumérica		A	A							P	P
Elemental	Constante	Alfabética		A	A	A							P
Elemental	Constante	Figurativa	SPACES	A	A	A							P
Elemental	Constante	Figurativa	HIGH-VALUES LOW-VALUES	A	A	A							P
Elemental	Constante	Figurativa	ZEROES	A	A		N	N	N	N	N	P	P
Elemental	Constante	Numérica		A	A		N	N	N	N	N	P	P
Elemental	Alfanumérico de editing			A	A								P
Elemental	Número editing			A	A								P
Elemental	Numérico	DISPLAY		A	A*		N	N	N	N	N	P	P
Elemental	Numérico	COMP		A	A*		N	N	N	N	N	P	P
Elemental	Numérico	COMP-3		A	A*		N	N	N	N	N	P	P
Elemental	Numérico	COMP-1					N	N	N	N	N	P	
Elemental	Numérico	COMP-2					N	N	N	N	N	P	

* Sólo números enteros

Move permitida (green cells)

Move no permitida (pink cells)

ción; si estas últimas no se respetan, el Compilador las presenta en cualquier caso como errores. Recurrir a esta sencilla costumbre puede ahorrar inútiles pérdidas de tiempo dedicadas a la búsqueda de errores que son difícilmente diagnosticables.

El siguiente ejemplo ilustra el caso en que una descripción errónea de un dato numérico comporta el resultado absurdo – 1 + 1 = 2.

```
01 CONTADOR          PIC 9.
–
–
–
–
–
PROCEDURE DIVISION.
CUENTA.
        MOVE – 1 TO CONTADOR.
        ADD    1 TO CONTADOR.
```

Como CONTADOR se ha descrito sin signo (sin el símbolo S), la transferencia de – 1 en el CONTADOR comporta la pérdida del signo algebraico; el contenido de CONTADOR después de la MOVE es 1 y no – 1. De esto resulta que el valor conseguido sumando 1 es 2 y no 0.

En la tabla de la pág. 1039 se han indicado sintéticamente todas las posibles combinaciones de la MOVE alfanuméricas y numéricas. En la tabla se ha utilizado el símbolo A para indicar una MOVE alfanumérica, N para una MOVE numérica y P para una MOVE por prospecto.

La instrucción Inspect

En el curso de un programa se puede tener la necesidad de entrar en el contenido de un campo, por ejemplo, para analizar el número de caracteres que no sean blank que contiene, o para verificar si en una posición cualquiera está presente una determinada combinación de caracteres. Exigencias de este tipo no pueden satisfacerse utilizando las instrucciones vistas hasta ahora; debe utilizarse una instrucción específica, la INSPECT.

En su formato más general, la instrucción INSPECT puede contar y sustituir las repeticiones de un grupo de uno o más caracteres presentes en un determinado campo.

Sin embargo, las dos funciones (contado y sustitución) pueden obtenerse separadamente utilizando dos subformatos.

La función de contado. Para efectuar la función de contado, la instrucción tiene necesidad de saber

1 / qué campo numérico utilizar como contador **(contador)**

2 / el nombre del campo que contiene la cadena de caracteres a analizar **(cadena)**

3 / qué debe contar **(combinación).**

El formato de la instrucción se indica en la tabla de la página siguiente, en la que se utiliza generalmente la palabra «combinación» para comprender la combinación de caracteres que debe buscarse en el ámbito de la cadena que se está analizando.

En realidad, para poder trabajar efectivamente, la instrucción debe saber también en qué modalidad se realiza la búsqueda.

La cláusula ALL. Considérese por ejemplo el caso en que se quiere conocer el número de caracteres A presentes en el campo

```
01 COLOR        PIC X(20) VALUE 'AMARILLO'.
```

El contado puede realizarse utilizando como contador el campo

```
01 CONT         PIC S9(5) COMP VALUE 0.
```

Debe tenerse en cuenta que durante la operación de contado, la instrucción INSPECT incrementa en 1 el valor del contador para cada combinación válida encontrada. Por tanto es necesario inicializar el contador a cero, a menos que no se desee un contado total al frente de más cadenas analizadas.

En el caso examinado, la instrucción INSPECT tendrá el siguiente formato:

```
INSPECT COLOR TALLYING CONT
                FOR ALL 'A'.
```

En este caso, la «combinación» necesaria es ALL 'A', o bien «todas las A presentes en la cadena». Después de la ejecución, el resultado será 3, memorizado en el campo CONT.

Obsérvese que se habría podido obtener el mismo resultado con las instrucciones

```
MOVE 'A' TO FLAG.
INSPECT COLOR TALLYING CONT
                FOR ALL FLAG.
```

FORMATO DE LA INSTRUCCION INSPECT PARA CONTADO

INSPECT cadena TALLYING contador FOR combinación.

donde FLAG es el campo de la WORKING-STORAGE SECTION

```
01 FLAG         PIC X.
```

La INSPECT puede utilizarse para buscar simultáneamente en el mismo campo más combinaciones de caracteres.
Considerando el campo COLOR anteriormente definido, supongamos que se desea contar simultáneamente cuantos grupos AMA, R, ILLO hay en el campo COLOR. Entonces puede procederse así:

```
01 COLOR        PIC X(20) VALUE
                          'AMARILLO'.
01 CUEN-AMA     PIC 9(2)  COMP VALUE 0.
01 CUEN-R       PIC 9(2)  COMP VALUE 0.
01 CUEN-ILLO    PIC 9(2)  COMP VALUE 0.
PROCEDURE DIVISION.
CUENTA.
     INSPECT COLOR
             TALLYING
             CUEN-AMA  FOR ALL 'AMA',
             CUEN-R    FOR ALL 'R',
             CUEN-ILLO FOR ALL 'ILLO'.
```

Los valores de los contadores después de la operación serán

```
CUEN-AMA  = 1
CUEN-R    = 1
CUEN-ILLO = 1
```

La cláusula LEADING. Utilizando la cláusula LEADING, la instrucción INSPECT permite verificar si una cierta combinación de caracteres está en las primeras posiciones del campo analizado. O sea, si se quiere saber si el contenido del campo COLOR empieza con una combinación de caracteres iguales a AM, se escribirá

```
INSPECT COLOR
        TALLYING CONTADOR
        FOR LEADING 'AM'.
```

Como en el caso en examen la palabra AMARILLO empieza con el grupo de caracteres 'AM', la instrucción incrementará en uno el campo CONTADOR. En cambio, si se hubiese escrito

```
INSPECT COLOR
        TALLYING CONTADOR
        FOR LEADING 'MAR'.
```

el contenido de CONTADOR habría quedado sin variación, puesto que el grupo MAR, también presente en la palabra AMARILLO, no constituye su parte inicial.

La cláusula CHARACTERS. Esta cláusula permite conocer el número de caracteres que constituyen un determinado campo; el valor proporcionado por la instrucción comprende también los eventuales blanks presentes. Es decir, la instrucción

```
INSPECT COLOR
        TALLYING CONTADOR
        FOR CHARACTERS.
```

restituye en contador el valor 20, o sea el número de caracteres declarados en la PICTURE del campo COLOR, y no 8, que son los caracteres diferentes de blank que constituyen la palabra AMARILLO.

La cláusula BEFORE. En todas las cláusulas del INSPECT vistas hasta ahora, el campo a analizar siempre se explora a partir del primer carácter de la izquierda y hacia la derecha.
Esta limitación puede eliminarse utilizando dos cláusulas que permiten efectuar todas las búsquedas descritas antes partiendo de una posición cualquiera de la cadena examinada.
El programador no fija de manera rígida este punto de partida, sino que está ligado al contenido del campo.
Es decir, es posible utilizar como punto de partida la posición de la cadena en que se ha verificado la existencia de una determinada combinación de caracteres.
Seguramente el lector ha tenido la sensación de una escasa utilidad de la cláusula CHARACTERS anteriormente descrita. Sin embargo, si

EJEMPLO DE USO DE LA CLAUSULA BEFORE

```
*
 01 ARTICULO.
    05 SERIE-NUMERO              PIC X(10).
    05 DESCRIPCION               PIC X(20).
    05 .........
    05 .........
    .
    .
    .
 01 K-CAR-SERIE                  PIC S9(2) COMP VALUE 0.
    88 TRES-CARACTERES VALUE 3.
    88 DOS-CARACTERES  VALUE 2.
    88 UN-CARACTER     VALUE 1.
*
*
*
 PROCEDURE DIVISION.
 INICIO.
  .
  .
  .
  .
 VER SERIE.
     INSPECT SERIE-NUMERO.
             TALLYING K-CAR-SERIE FOR CHARACTERS BEFORE '/'.
     IF TRES-CARACTERES
        PERFORM PROCESA-SERIE-3.
     IF DOS-CARACTERES
        PERFORM PROCESA-SERIE-2.
     IF UN-CARACTER
        PERFORM PROCESA-SERIE-1.
  .
  .
  .
```

esta cláusula se usa conjuntamente con la BEFORE se dispone de aplicaciones más interesantes. Efectivamente, considérese el caso en el que en un programa para la gestión de un almacén, el código de serie de cada artículo puede estar compuesto por un número cualquiera de caracteres separados simplemente por el caracter / del número de artículo, como

```
ABC / 1234
 ZX / 789
  U / 045
```

Supongamos que se quiera procesar de manera diferente cada artículo según que la serie esté constituida por 1, 2, 3,..., N caracteres.
En este caso, la única manera posible para conocer exactamente a qué serie pertenece el artículo es proceder como se indica en el listado de arriba.
Volviendo al ejemplo ilustrado al hablar del primer formato de la cláusula CHARACTERS, se habrían podido contar los caracteres que constituyen la palabra AMARILLO utilizando indiferentemente uno de los dos formatos siguientes:

```
INSPECT COLOR
    TALLYING CONTADOR
    FOR CHARACTERS BEFORE ' '
INSPECT COLOR
    TALLYING CONTADOR
    FOR CHARACTERS BEFORE
    SPACES.
```

En ambos casos, el campo CONTADOR habría contenido, después de la ejecución, el valor 8. Puede obsevarse como en el segundo formato se ha utilizado la constante figurativa SPACES en lugar del carácter ' '. Esto es posible para cualquier formato de la INSPECT. Sin embargo, téngase presente que en este contexto SPACES equivale a un solo espacio en blanco.

La cláusula AFTER. Mientras que en la cláusula BEFORE la INSPECT del campo empieza por

el primer carácter a la izquierda y se para cuando se comprueba la existencia del carácter declarado en la BEFORE, con la cláusula AFTER se tiene el comportamiento opuesto. El campo se explora a partir de la izquierda para verificar la existencia del carácter declarado. Efectuada la verificación empieza el control de las otras condiciones indicadas en la instrucción.
Considerando siempre el caso del campo

```
01 COLOR     PIC X(20) VALUE 'AMARILLO'.
```

y si se quiere conocer el número de caracteres que siguen a la primera L presente en el campo, entonces puede utilizarse la instrucción

```
        INSPECT COLOR
                TALLYING CONTADOR
                FOR CHARACTERS AFTER 'L'.
```

Recuérdese que las cláusulas BEFORE y AFTER no pueden existir simultáneamente en la misma instrucción.
Si por ejemplo se quisiese conocer el número de caracteres comprendidos entre el grupo AMA y el grupo LO, la instrucción

```
        INSPECT COLOR
                TALLYING CONTADOR
                FOR CHARACTERS
                    BEFORE    'LO'
                    AFTER     'AMA'
```

es errónea.
Para satisfacer la misma exigencia es necesario recurrir a más instrucciones INSPECT oportunamente dispuestas y calcular el valor buscado como diferencia entre los valores obtenidos.
En conclusión, el formato general de la instrucción INSPECT usada para contar caracteres se indica en la tabla de abajo.

La función de sustitución. Para poder efectuar la sustitución de una parte de los caracteres presentes en un campo, la INSPECT tiene necesidad de conocer

1 / qué campo debe analizar
2 / qué debe sustituir
3 / con qué.

El formato correspondiente a esta segunda función es el siguiente

```
INSPECT campo REPLACING combinación-existente
                             BY
                        combinación-nueva
```

Todo esto puede traducirse en un lenguaje corriente como sigue:
«inspecciona (INSPECT) el contenido de **campo** sustituyendo (REPLACING) la combinación de caracteres **combinación-existente** con (BY) la nueva combinación **combinación-nueva**».
De manera completamente análoga al formato correspondiente a la función de contado, la locución «combinación-existente» también comprende sintéticamente las cláusulas para la descripción de los caracteres a sustituir, como se ha descrito en el formato general indicado en la tabla de la página siguiente.
Para mayor claridad examinaremos ahora algu-

FORMATO GENERAL DE LA INSTRUCCION INSPECT CON FUNCIONES DE CONTADO

```
                                          { { ALL     } { nombre-de-dato        } }
                                          { {         } { constante             } }
                                          { { LEADING } { constante-figurativa  } }
INSPECT campo TALLYING contador FOR       {                                       }
                                          {   CHARACTERS                          }

      [ { BEFORE }           { nombre-de-dato-1 } ]
      [ {        }  INITIAL  {                  } ]
      [ { AFTER  }           { constante-1      } ]
```

FORMATO GENERAL DE LA INSTRUCCION INSPECT CON FUNCIONES DE SUSTITUCION

```
                                ⎧ ⎧ ALL     ⎫        ⎧ nombre-de-campo ⎫ ⎫
                                ⎪ ⎨ LEADING ⎬        ⎨                 ⎬ ⎪
INSPECT campo REPLACING         ⎨ ⎩ FIRST   ⎭        ⎩ constante       ⎭ ⎬
                                ⎪                                        ⎪
                                ⎩   CHARACTERS                           ⎭

                BY                ⎧ nombre-de-campo-1 ⎫
                                  ⎩ constante-1       ⎭

                               ⎡⎧ BEFORE ⎫            ⎧ nombre-de-campo-2 ⎫⎤
                               ⎣⎩ AFTER  ⎭  INITIAL   ⎩ constante-2       ⎭⎦
```

nos ejemplos de INSPECT considerando el esquema del contenido del campo analizado después de la ejecución de la instrucción.
Considérese el campo CIFRAS-EN-LETRAS, que tiene la descripción

```
01 CIFRAS-EN-LETRAS     PIC X(30)
   VALUE 'MILSETECIENTOSVEINTITRES'.
```

La configuración inicial de los caracteres en el interior del campo y las diversas configuraciones asumidas por el campo después de la ejecución de algunas instrucciones INSPECT se indican en la figura de arriba de la pág. 1045.

Formato general de la instrucción INSPECT. Como hemos anticipado al principio de la explicación de la INSPECT, usando esta instrucción se pueden efectuar simultáneamente tanto operaciones de contado como operaciones de sustitución. Además sabemos que podemos efectuar más operaciones de tipo diferente sobre el mismo campo.
Supóngase, por ejemplo, que se quiere efectuar en el campo inicial escrito en la figura de arriba de la página siguiente:

1 / el contado de todos los caracteres diferentes de blank contenidos en el campo CIFRAS-EN-LETRAS, utilizando K-CARACTERES como contador
2 / el contado de todos los caracteres 'T' utilizando K-T como contador
3 / la sustitución del primer grupo 'CIENTOS' por la cadena 'WWWWWWW'
4 / la sustitución de todas las letras 'T' que siguen al grupo 'SETE' por 'N'.

La instrucción INSPECT tendrá entonces la forma que se indica a continuación:

```
INSPECT CIFRAS-EN-LETRAS
         TALLYING K-CARACTERES
                  FOR CHARACTERS
                      BEFORE ' '
                  K-T
                  FOR ALL 'T'
         REPLACING FIRST 'CIENTOS'
                   BY 'WWWWWWW'
                  ALL 'T' BY 'N'
                      AFTER 'SETE'.
```

Después de la ejecución de la instrucción se tendrá

```
K-CARACTERES  = 24
K-T           =  7
```

mientras que la configuración del campo será

M	I	L	S	E	T	E	W	W	W	W	W	W	W	V	E	I	N	T	I	T	R	E	S								

La instrucción INSPECT, cuyo formato completo se indica en la tabla de abajo de la página siguiente, es aplicable también a campos numéricos, aunque con USAGE DISPLAY, implícita o explícita.

EFECTOS DE ALGUNAS INSTRUCCIONES INSPECT

CONFIGURACION INICIAL DEL CAMPO CIFRAS-EN-LETRAS

M	I	L	S	E	T	E	C	I	E	N	T	O	S	V	E	I	N	T	I	T	R	E	S						

INSPECT CIFRAS-EN-LETRAS REPLACING CHARACTERS BY '*'.

*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

INSPECT CIFRAS-EN-LETRAS REPLACING LEADING 'MIL' BY '– – –'.

–	–	–	S	E	T	E	C	I	E	N	T	O	S	V	E	I	N	T	I	T	R	E	S						

INSPECT CIFRAS-EN-LETRAS REPLACING ALL ' ' BY '*'.

M	I	L	S	E	T	E	C	I	E	N	T	O	S	V	E	I	N	T	I	T	R	E	S	*	*	*	*	*	*

INSPECT CIFRAS-EN-LETRAS REPLACING FIRST 'E' BY 'A'.

M	I	L	S	A	T	E	C	I	E	N	T	O	S	V	E	I	N	T	I	T	R	E	S						

INSPECT CIFRAS-EN-LETRAS REPLACING ALL 'T' BY 'S'
ALL 'S' BY '8'
BEFORE 'VEINTE'

M	I	L	8	E	S	E	C	I	E	N	S	0	8	V	E	I	N	T	I	T	R	E	S						

FORMATO GENERAL DE LA INSTRUCCION INSPECT

```
INSPECT campo TALLYING contador FOR { { ALL     }  { nombre-de-dato-1 } }
                                    { { LEADING }  { constante-1      } }
                                    {   FIRST                           }

        [ { BEFORE }  INITIAL  { nombre-de-dato-2 } ]
        [ { AFTER  }           { constante-2      } ]

REPLACING  { { ALL        }  { nombre-de-dato-3 } }
           { { LEADING    }  {                  } }
           { { CHARACTERS }  { constante-3      } }
           {                                      }
           {   CHARACTERS                         }

     BY    { nombre-de-dato-4 }
           { constante-4      }

        [ { BEFORE }  INITIAL  { nombre-de-dato-5 } ]
        [ { AFTER  }           { constante-5      } ]
```

Palabras subrayadas: INSPECT, TALLYING, FOR, ALL, LEADING, FIRST, BEFORE, AFTER, REPLACING, CHARACTERS, BY.

La instrucción STRING

Para completar el examen de las instrucciones orientadas a la manipulación de los datos, se analizarán ahora dos importantes instrucciones: la STRING y la UNSTRING, que en cierto sentido reúnen las posibilidades de la MOVE y de la INSPECT. Sin embargo, debe observarse que la MOVE puede transferir datos de un campo de partida a uno de llegada siempre que ambos estén descritos en la DATA DIVISION.

La instrucción MOVE o no entra del todo en el contenido del campo (MOVE alfanumérica) o bien cuando lo hace (MOVE numérica) sólo es para respetar la tipología del campo de llegada. Es decir, para la MOVE, el valor del dato transferido es completamente transparente y, en general, sólo es posible desplazar una parte del dato, a menos que el campo esté oportunamente subdefinido.

Por ejemplo, si en un determinado instante el campo CADENA

```
01 CADENA.
   05 SUBCADENA-1          PIC X(3).
   05 SUBCADENA-2          PIC X(5).
   05 SUBCADENA-3.
      10 SUBCADENA-31      PIC X.
      10 SUBCADENA-32      PIC XX.
      10 SUBCADENA-33      PIC X(6).
```

contuviese el valor 'EJEMPLO DE MOVE', el contenido de los diferentes subcampos sería

```
SUBCADENA-1  = 'EJE'
SUBCADENA-2  = 'MPLO  '
SUBCADENA-31 = 'D'
SUBCADENA-32 = 'E '
SUBCADENA-33 = 'MOVE  '
```

Entonces no sería posible desplazar por ejemplo la preposición 'DE', puesto que los caracteres componentes pertenecen a dos subcampos diferentes.

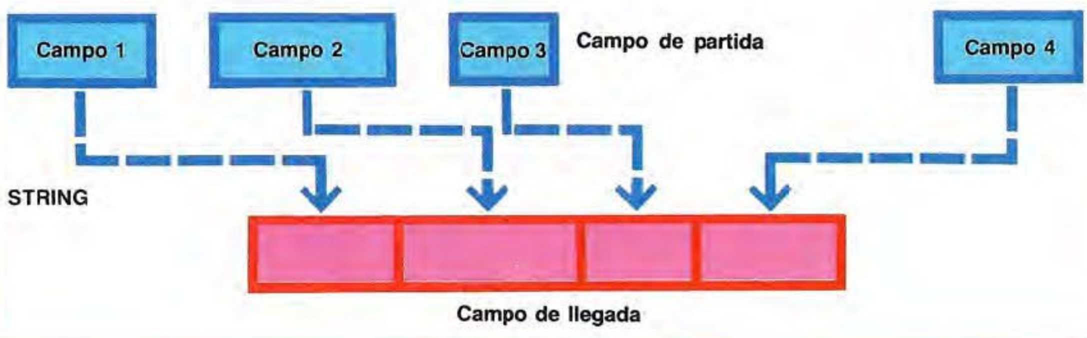
Contrariamente al comportamiento de la instrucción MOVE, la instrucción INSPECT prescinde de eventuales subdescripciones del campo analizado, puede inspeccionar el contenido y, eventualmente, alterarlo.

El único modo para analizar el contenido de un campo y efectuar el desplazamiento de algunas de sus partes al producirse oportunas condiciones, es el de utilizar las instrucciones STRING y UNSTRING. La instrucción STRING permite transferir el contenido completo o parcial de uno o más campos en un único campo de llegada, disponiéndolos uno a continuación de otro según el orden declarado por el programador (ver el esquema indicado abajo). Supóngase que se tenga que controlar si una fecha tecleada en la forma DD/MM/AA (día/mes/año) sea antecedente o no a la fecha de proceso.

Mediante la instrucción ACCEPT es posible tomar del calendario del procesador la fecha del día. Ésta la proporciona el campo declarado en la forma AAMMDD (año/mes/día). Por otra parte, esta forma es la más idónea para el control de secuencias de las fechas: que el 13 de diciembre de 1983 siga al 11 de noviembre de 1983 es fácilmente verificable porque 831213 (representación del 13 de diciembre de 1983) es mayor que 831111 (representación de 11 de noviembre de 1983). Por tanto, volviendo a considerar el ejemplo, es evidente que la fecha indicada en la forma DD/MM/AA debe convertirse oportunamente para permitir su control.

El problema puede resolverse con el programa indicado en el listado de la página siguiente.

ACCION DE LA INSTRUCCION STRING



CONVERSION DE UNA FECHA

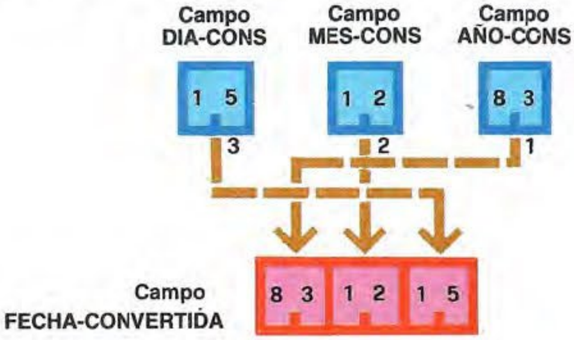
```
 01 FECHA-CONSOLA.
     05 DIA-CONS                  PIC 9(2).
     05 FILLER                    PIC X.
     05 MES-CONS                  PIC 9(2).
     05 FILLER                    PIC X.
     05 AÑO-CONS                  PIC 9(2).
     05 FILLER                    PIC X.
*
*
 01 FECHA-PROCESO                 PIC 9(6).
 01 FECHA-PROC       REDEFINES FECHA-PROCESO.
     05 AÑO-PROC                  PIC 9(2).
     05 MES-PROC                  PIC 9(2).
     05 DIA-PROC                  PIC 9(2).
*
*
 01 FECHA CONVERTIDA.
     05 AÑO-CONV                  PIC 9(2).
     05 MES-CONV                  PIC 9(2).
     05 DIA-CONV                  PIC 9(2).
*
*
 PROCEDURE DIVISION.
 ACEPTA-FECHAS.
     DISPLAY '** TECLEAR FECHA (DD/MM/AA)'
             UPON CONSOLE
     ACCEPT FECHA-CONSOLA FROM CONSOLE.
     STRING AÑO-CONS
             MES-CONS
             DIA-CONS
                       DELIMITED BY SIZE
                       INTO     FECHA-CONVERTIDA.
     ACCEPT FECHA-PROCESO FROM FECHAS.
     IF DATA CONVERTIDA IS GREATER THAN FECHA-PROCESO
        PERFORM .........
        .................
        .................
*
*
```

La instrucción

```
STRING AÑO-CONS
       MES-CONS
       DIA-CONS
       DELIMITED BY SIZE
INTO FECHA-CONVERTIDA
```

activa las operaciones esquematizadas en el gráfico de la pág. 1048, donde se supone que en la consola se ha tecleado la fecha 15/12/83.
La instrucción, en definitiva, transfiere (STRING) en el campo receptor (INTO FECHA-CONVERTIDA) y en el orden indicado todo el contenido (DELIMITED BY SIZE) de los campos AÑO-CONS, MES-CONS, DIA-CONS.
La cláusula DELIMITED BY SIZE indica que la transferencia contempla las máximas dimensiones del campo a transferir, tal como se han declarado en DATA DIVISION. Sin embargo, los tres campos transferidos se habían declarado como de dos caracteres cada uno: PIC 9(2).
Es evidente que, en este caso, para obtener el mismo resultado habrían podido usarse tres MOVE en lugar de la STRING.
La flexibilidad de empleo de la STRING puede aprovecharse al máximo conociendo todas las cláusulas que prevé la instrucción.
Los siguientes ejemplos dan una idea del potencial de este verbo, que actúa sobre campos alfanuméricos y puede crear en un campo de llegada una composición de cadena de caracteres tomados por más campos de partida.
Los caracteres a tomar se identifican en base a uno o más caracteres declarados como delimitadores. Por ejemplo, considérese el campo

CONVERSION DEL FORMATO FECHA CON LA INSTRUCCION STRING



Campos DIA-CONS, MES-CONS, AÑO-CONS después de la ejecución de la instrucción ACCEPT

Operaciones efectuadas por la STRING. Los números 1, 2, 3 indican la secuencia seguida para efectuar la transferencia

Campo FECHA-CONVERTIDA después de la ejecución de la STRING

```
    01 PARTIDA          PIC X(4).
```

cuyo contenido, en un cierto momento, valga

ABCD

Se desea componer en el campo

```
    01 LLEGADA          PIC X(40).
```

una cadena de caracteres que contiene:

1 / la constante alfanumérica: EJEMPLO DE CADENA
2 / un blank
3 / el carácter: (dos puntos)
4 / un blank
5 / todas las letras que preceden a la letra C en el contenido del campo PARTIDA.

La instrucción que permite obtener lo especificado es la siguiente:

```
STRING 'EJEMPLO DE STRING' DELIMITED
                           BY SIZE
        ' '                DELIMITED
                           BY SIZE
        ':'                DELIMITED
                           BY SIZE
        SPACE              DELIMITED
                           BY SIZE
        PARTIDA            DELIMITED
INTO                       BY 'C'
        LLEGADA.
```

En este caso, la composición del campo de llegada empieza por el primer carácter de la izquierda, pero es posible establecer un punto diferente de partida utilizando un puntero. Por ejemplo, la cadena presentada anteriormente puede componerse a partir de la séptima posición en el campo LLEGADA, utilizando las siguientes instrucciones.

```
MOVE 7 TO POSICION.
STRING 'EJEMPLO DI STRING'
        ' '
        ':'
        SPACE       DELIMITED BY SIZE
        PARTIDA     DELIMITED BY 'C'
INTO
        LLEGADA  WITH POINTER POSICION
ON OVERFLOW
        DISPLAY 'CAMPO LLEGADA
                 INSUFICIENTE'
                 UPON CONSOLE.
```

En el ejemplo se ha insertado también la cláusula ON OVERFLOW, que permite controlar la eventualidad de que el campo de llegada no pueda contener todos los caracteres que se han declarado.
El formato general de la instrucción STRING se ha indicado en la parte de arriba de la página siguiente.

FORMATO GENERAL DE LA INSTRUCCION STRING

```
STRING   { nombre-de-dato-1 }  [ { nombre-de-dato-2 } ] ..........
         { constante-1      }  [ { constante-2      } ]

            DELIMITED BY       { nombre-de-dato-3 }
                               { constante-3      }
                               { SIZE             }

      [  { nombre-de-dato-4 }  [ { nombre-de-dato-5 } ] ..........  ]
      [  { constante-4      }  [ { constante-5      } ]             ]
      [                                                             ]
      [     DELIMITED BY       { nombre-de-dato-6 }                 ]
      [                        { constante-6      }                 ]
      [                        { SIZE             }                 ]

            INTO nombre-de-dato-7
                 [WITH POINTER nombre-de-dato-8]
                 [ON OVERFLOW frase-imperativa]
```

ACCION DE LA INSTRUCCION UNSTRING



La instrucción UNSTRING

Como puede intuirse, el verbo UNSTRING trabaja de manera opuesta a la de STRING. Permite tomar datos de un campo alfanumérico único y transferirlos a uno o más campos de llegada, según el esquema de la figura de arriba.
La identificación de los caracteres a tomar se efectúa con la misma lógica usada por la STRING, o bien especificando el carácter que debe entenderse como delimitador (DELIMITED BY...). Además, la UNSTRING está dotada de cláusulas de las cuales no existe correspondencia en la STRING, como se ve en el formato general de la tabla de arriba de la pág. 1050.

La cláusula ALL permite hacer interpretar al Compilador un grupo de caracteres iguales como un único delimitador. De este modo, por ejemplo, la

DELIMITED BY ALL 'L'

hace que, al analizar la cadena

BELLEZA

el grupo 'LL' se entienda como un único delimitador. Si no se especificase la cláusula ALL sólo se trataría como delimitador la primera L, mien-

FORMATO GENERAL DE LA INSTRUCCION UNSTRING

```
UNSTRING nombre-de-dato-1

    [ DELIMITED BY ALL { nombre-de-dato-2 }
                       { constante-1      }

          [ OR [ALL] { nombre-de-dato-3 } ] ]
                     { constante-2      }

INTO  nombre-de-dato-4

      [DELIMITER IN     nombre-de-dato-5]
      [COUNT IN         nombre-de-dato-6]

    [ nombre-de-dato-7
                                          ]   ..........
      [DELIMITER IN     nombre-de-dato-8]
      [COUNT IN         nombre-de-dato-9] ]

[WITH POINTER  nombre-de-dato-10]
[TALLYING IN   nombre-de-dato-11]
[ON OVERFLOW  frase-imperativa].
```

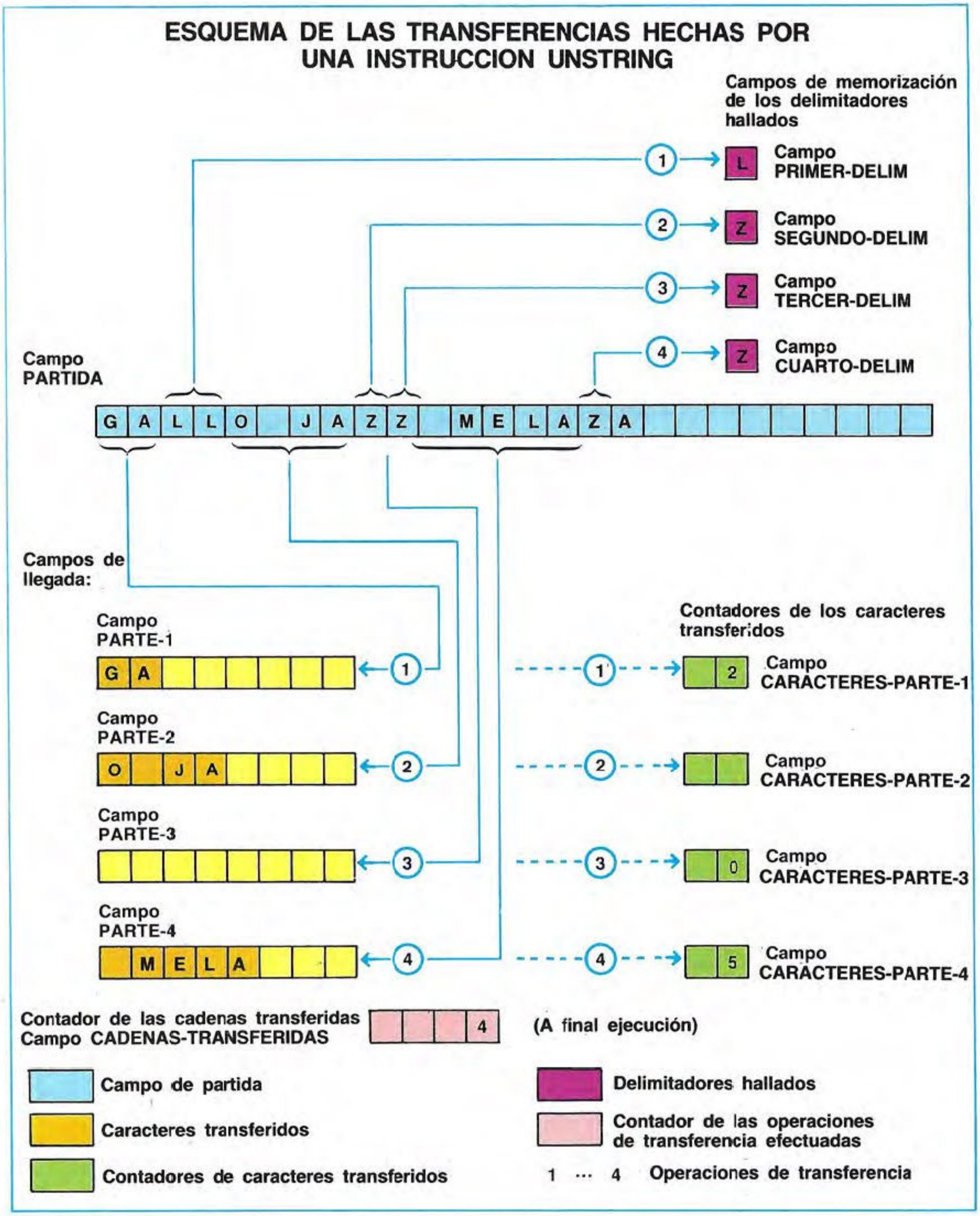
EJEMPLO DE USO DE LA INSTRUCCION UNSTRING

```
MOVE 'GALLO JAZZ MELAZA' TO PARTIDA.
UNSTRING PARTIDA
         DELIMITED BY Z,                         Definición de
                      ALL 'L'                    los delimitadores
INTO PARTE-1
     DELIMITER   IN   PRIMER-DELIM               Primera transferencia
     COUNT       IN   CARACTERES-PARTE-1
  *
     PARTE-2
     DELIMITER   IN   SEGUNDO-DELIM              Segunda transferencia
     COUNT       IN   CARACTERES-PARTE-2
  *
     PARTE-3
     DELIMITER   IN   TERCER-DELIM               Tercera transferencia
     COUNT       IN   CARACTERES-PARTE-3
  *
     PARTE-4
     DELIMITER   IN   CUARTO-DELIM               Cuarta transferencia
     COUNT       IN   CARACTERES-PARTE-4
  *
     TALLYING CADENAS-TRANSFERIDAS               Contador de las operaciones
     ON OVERFLOW                                 de transferencia efectuadas
        DISPLAY 'OVERFLOW DE UNSTRING'
              UPON CONSOLE.
```

tras que la segunda se entendería como perteneciente a la siguiente cadena.
El significado de las diversas cláusulas de la instrucción se ilustra abajo, donde se ha representado gráficamente la serie de acciones realizadas por la instrucción UNSTRING indicada en la tabla de abajo de la página anterior.

Verbos de control

El tratamiento desarrollado hasta ahora ha proporcionado al lector el conocimiento de todas las operaciones que pueden efectuarse sobre datos, desde las operaciones de INPUT/OUTPUT para el tratamiento de files secuenciales



tanto en lectura como en escritura, hasta las orientadas a la manipulación de los datos en memoria. Las instrucciones analizadas hasta ahora constituyen en cierto sentido el armazón del programa Cobol; sin embargo, todavía no podemos organizar un programa.

Efectivamente, un programa es una estructura lógica constituida por una secuencia de decisiones y acciones secuenciales, emprendidas frente a situaciones razonablemente previsibles en la fase de análisis.

El programador debe poder alterar en cualquier momento el flujo secuencial de las instrucciones al producirse determinadas condiciones que necesitan tratamientos diversificados de los datos. A tal fin, el Cobol dispone de un conjunto de instrucciones que sirven para el control de las secuencias de las operaciones realizadas. Estas instrucciones ya se han usado en algunos ejemplos presentados, contando con la intuición del lector para su interpretación, facilitada además por la forma sintáctica común al lenguaje Basic ya examinado.

Este capítulo se dedicará al examen de las instrucciones de control en su forma más general.

La proposición IF

Mediante esta proposición es posible evaluar la verificación de una determinada circunstancia y, en función del resultado del análisis, emprender dos acciones de tipo diferente.

Conceptualmente, la proposición IF equivale a la siguiente frase: «si (IF) se verifica una determinada condición, entonces (THEN) realiza la ACCION-A, de otro modo (ELSE) realiza la ACCION-B». Utilizando el verbo PERFORM cuya función, ya indicada, se comentará más adelante, se tendrá:

```
IF   condición
     THEN
         PERFORM ACCION-A
     ELSE
         PERFORM ACCION-B.
```

donde ACCION-A y ACCION-B son dos nombres de párrafos o de secciones que contienen secuencias diferentes de instrucciones.

Obsérvese que el adverbio THEN no forma parte del ANS Cobol y que sólo se utilizará en los primeros ejemplos para hacer más evidente la construcción lógica de la frase entera.

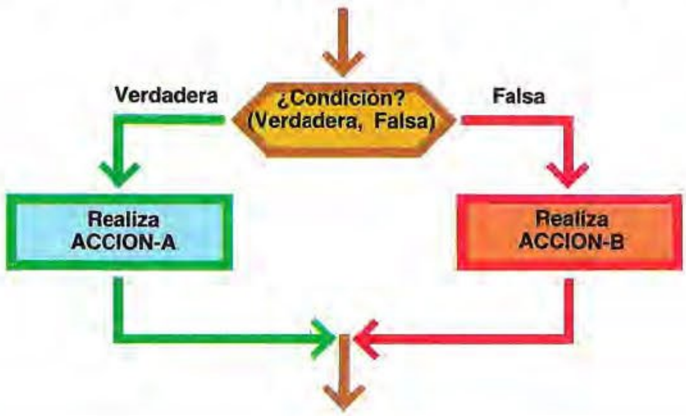
Después de los primeros ejemplos ya no se hará uso de este adverbio, acudiendo así a las especificaciones del Cobol estándar.

La representación gráfica de la frase condicional se indica en la figura de abajo.

Además se quiere llamar la atención del lector sobre la importancia que reviste el punto (.) en el ámbito de una frase condicional: omitir un punto o insertar uno erróneamente puede inducir al Compilador a una interpretación de la frase diferente de la entendida por el programador. En este caso, la salida del proceso puede ser imprevisible (terminación por error, bucles infinitos, desplazamiento más allá del espacio de memoria asignado, etc.), según el contexto en que está inserta la frase.

El formato general de la instrucción IF se ha indicado en la tabla de la página siguiente, pero antes de analizarla, es importante subrayar el

DIAGRAMA DE FLUJO DE UNA INSTRUCCION IF



FORMATO GENERAL DE LA INSTRUCCION IF

```
                           ( NEXT SENTENCE      )   [        ( NEXT SENTENCE      ) ]
IF condición   [THEN]     {                      }   [ ELSE  {                      } ]
                           ( frase-imperativa-1 )   [        ( frase-imperativa-2 ) ]
```

hecho de que, análogamente a lo que sucede en el lenguaje humano, las frases condicionales constituyen puntos de referencia para la comprensión del flujo de un programa.

Al leer un programa escrito por uno mismo o por otra persona, el primer esfuerzo a realizar es el de identificar los puntos de decisión, los puntos de derivación y las correspondientes condiciones analizadas.

En una primera aproximación de este tipo tiene muy poca importancia entrar en el detalle de cada instrucción sencilla si antes no se ha identificado el contexto lógico en que trabaja la instrucción. No es exagerado afirmar que **leer un programa equivale a leer los IF que contiene**. Se deduce que los puntos de decisión del programa deben detallarse de manera que resalten desde la primera ojeada al listado.

El Cobol permite escribir la mayor parte de las instrucciones, siempre respetando las reglas sintácticas que las controlan, de manera que la disposición de los operandos ya dé una sensación de su función lógica. El formato de la instrucción IF resulta de difícil interpretación si se emplea como se ha descrito en la página anterior. Por tanto, se aconseja vivamente utilizarlo siempre con las cláusulas detalladas de la siguiente manera:

```
IF condición

   [THEN]   { NEXT SENTENCE      }
            { frase-imperativa-1 }

 [ ELSE     { NEXT SENTENCE      } ]
 [          { frase-imperativa-2 } ]
```

De esta manera se tiene una doble ventaja:

- resaltar la IF en el contexto del programa
- aislar visualmente las instrucciones que constituyen las dos alternativas del proceso.

Obsérvese a propósito el siguiente ejemplo, en el que las instrucciones utilizadas en la IF son sintácticamente correctas, pero cuya disposición impide extraer rápidamente el sentido de las operaciones efectuadas antes de las dos vías de la frase condicional.

```
IF TIPO-FICHA = 'A' MOVE FICHA
TO DISPONIBLE-FICHA
COMPUTE SALDO = HABER-DEBE
ADD SALDO TO SALDO-TOTAL
ADD 1 TO FICHAS-LEIDAS
GO TO LEE-FICHAS ELSE
ADD 1 TO FICHAS-LEIDAS
ADD 1 TO FICHAS-DESCARTADAS
GO TO LEE-FICHAS.
```

A continuación se indica el mismo ejemplo detallado en la forma aconsejada:

```
IF TIPO-FICHA = 'A'
   MOVE FICHA TO DISPONIBLE-FICHA
   COMPUTE SALDO = HABER-DEBE
   ADD SALDO       TO SALDO-TOTAL
   ADD 1           TO FICHAS-LEIDAS
   GO TO LEE-FICHAS

ELSE
   ADD 1          TO FICHAS-LEIDAS
                     FICHAS-DESCARTADAS
   GO TO LEE-FICHAS.
```

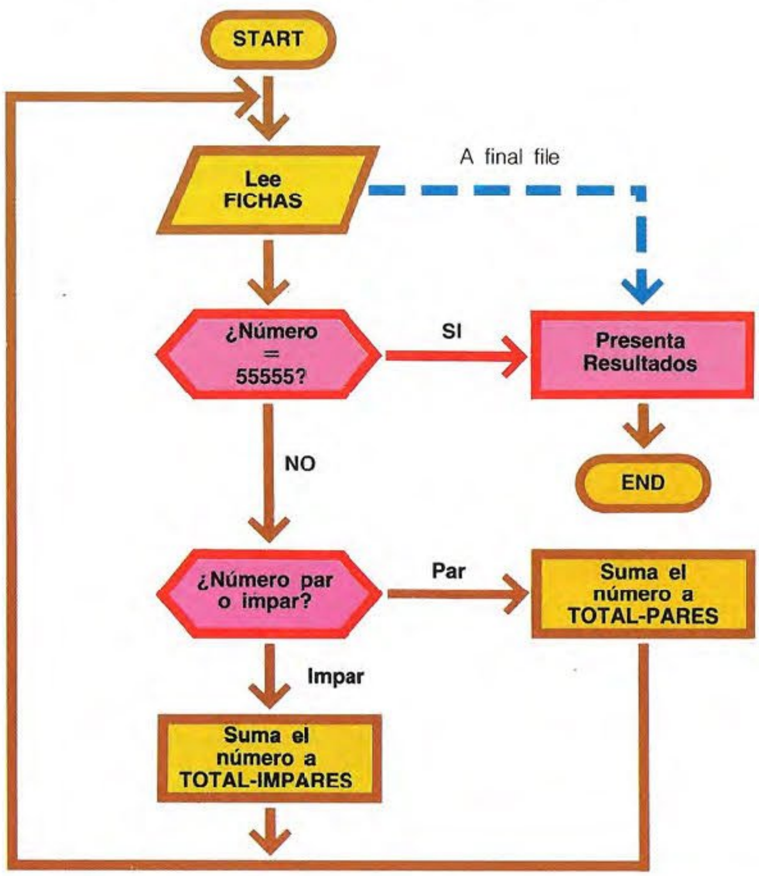
Del análisis de este detalle puede obtenerse fácilmente el diagrama de flujo y, por tanto, la comprensión de las funciones realizadas.

Lógica de ejecución de la instrucción IF. Antes de entrar en el detalle de los tipos de análisis que la proposición IF puede efectuar, es interesante conocer la dinámica según la cual se desarrolla el análisis en sí. Si la condición especificada inmediatamente después de la proposición IF resulta verdadera, se realiza la frase inmediatamente siguiente a dicha condicion, has-

ta encontrar la palabra ELSE o el primer punto. Después de la ejecución de la frase, el control pasa a la primera frase que sigue al primer punto. En cambio, si la condición resulta falsa, se realiza la frase inmediatamente siguiente a la proposición ELSE (si está presente) y el control pasa después a la primera instrucción que sigue al punto de cierre de la frase condicional.
Es oportuno aclarar la dinámica ilustrada con algunos ejemplos.
Se quiere calcular las sumas de los números enteros, pares e impares, perforados uno en cada ficha de un file de fichas; el proceso termina, y los resultados obtenidos deben presentarse en consola, apenas se encuentre en el file una ficha con el número 55555.
Es evidente que, para poder efectuar la suma de los números pares por una parte y la de los números impares por otra, debe reconocerse la naturaleza del número leído en la ficha.
Por otra parte, antes de procesar el número leído debe verificarse que el número no sea igual a 55555.
En una primera aproximación, el programa tendrá el diagrama de flujo ilustrado abajo.
Para evaluar si un número es par o impar puede dividirse por dos: si el resto de la división es igual a 0, el número es par, de otra forma es impar. Por tanto, el programa descrito puede detallarse como se ve en el listado de la página siguiente.
Leyéndolo puede observarse que si la condición NUMERO = 55555 resulta verdadera, se envía a ejecución la instrucción GO TO FIN-PROC, mientras que todas las veces que resulta falsa, el control pasa a la siguiente instrucción en el punto:

```
DIVIDE NUMERO BY 2
        GIVING COCIENTE
        REMAINDER RESTO.
```

EJEMPLO DE FRASES CONDICIONALES



APLICACION DE LA INSTRUCCION IF

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PRUEBA-IF.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. .............
 OBJECT-COMPUTER. .............
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FICHAS ASSIGN TO CARD-READER FICHAS.
*
*
*
 DATA DIVISION.
 FILE SECTION.
 FD FICHAS
    LABEL RECORD OMITTED.
 01 FICHA                       PIC X(80).
*
*
 WORKING-STORAGE SECTION.
 01 FICHA-WS.
    05 NUMERO                    PIC 9(5) COMP.
    05 FILLER                    PIC X(75).
*
 01 COCIENTE                     PIC 9(5)V9 COMP.
 01 RESTO                        PIC 9(2)   COMP.
 01 SUMA-PARES                   PIC 9(10)  COMP VALUE 0.
 01 SUMA-IMPARES                 PIC 9(10)  COMP VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 INICIO SECTION.
 ABRE-FILE.
    OPEN INPUT FICHAS.
 LEE-FICHA.
    READ FICHAS INTO FICHA-WS
                AT END
                GO TO FIN-PROC.
    IF NUMERO = 55555
       GO TO FIN-PROC.
    DIVIDE NUMERO BY 2
           GIVING COCIENTE
           REMAINDER RESTO.
    IF RESTO = 0
       ADD NUMERO TO SUMA-PARES
    ELSE
       ADD NUMERO TO SUMA-IMPARES.
    GO TO LEE-FICHA.
*
*
*
 FIN-PROC.
    DISPLAY '** SUMA NUMEROS PARES = ' SUMA-PARES
            UPON CONSOLE.
    DISPLAY ' '
            UPON CONSOLE.
    DISPLAY '** SUMA NUMEROS IMPARES = ' SUMA-IMPARES
            UPON CONSOLE.
 CIERRA-FILE.
    CLOSE FICHAS.
 FIN.
    DISPLAY '*** PROCESO TERMINADO ***'
            UPON CONSOLE.
    STOP RUN.
```

Después de la división del número por 2, el resto es igual a 0 o no lo es. En el caso en que resulte verdadera la primera hipótesis, el número (par) se suma a SUMA-PARES, de otra forma (ELSE) se suma a SUMA-IMPARES.
La frase condicional termina con el punto que se encuentra al final de la instrucción

ADD NUMERO TO SUMA-IMPARES.

Cualquiera que haya sido el camino seguido por el programa en la frase condicional, al final el control se cede siempre a la instrucción inmediatamente siguiente a dicha frase condicional. Terminado el análisis sobre el valor del resto (RESTO) y realizada la correspondiente instrucción, el programa vuelve al párrafo LEE-FICHA. Este ciclo se realiza iterativamente hasta que se termina el file (AT END) o hasta que se lee una ficha con el número 55555. A propósito de la importancia que reviste el punto en la exacta definición de una frase condicional, obsérvese que si, en el ejemplo, el segundo IF se hubiese escrito erróneamente de la manera

```
IF RESTO = 0
   ADD NUMERO TO SUMA-PARES
ELSE
   ADD NUMERO TO SUMA-IMPARES
GO TO LEE-FICHA.
```

(o bien si se hubiese omitido el punto en la instrucción ADD NUMERO TO SUMA-IMPARES) el Compilador habría interpretado el flujo de las operaciones en la forma esquematizada en la figura de la página siguiente.
El programa habría vuelto a leer las fichas sólo en el caso en que el número tratado anteriormente hubiese sido impar; al encontrar el primer número par, después de la suma del número en SUMA-PARES, el control habría pasado al párrafo FIN-PROC, con el consiguiente cierre del proceso.

La opción NEXT SENTENCE. La opción NEXT SENTENCE permite transferir el control de las operaciones a la primera instrucción inmediatamente siguiente al punto de cierre de la frase condicional.
Para aclarar completamente el concepto se volverá a considerar el IF del ejemplo anterior que se encuentra en primer lugar:

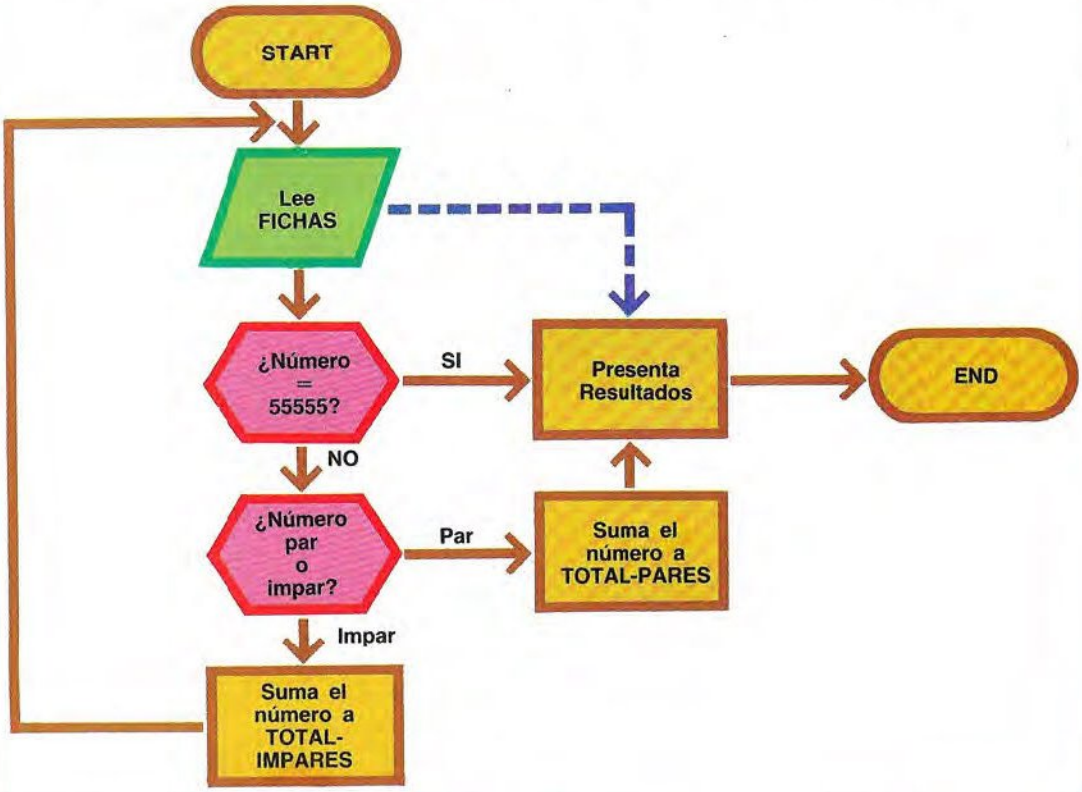
```
IF NUMERO = 55555
   GO TO FIN-PROC.
DIVIDE ....................
   ............................
```

Test numérico de un display de cristales líquidos (LCD, Liquid Cristal Display).



M. Wolf/Grazia Neri-Black Star

INTERPRETACION ERRONEA DE UNA FRASE CONDICIONAL



Se habría obtenido el mismo resultado con

```
IF NUMERO NOT = 55555
  NEX SENTENCE
ELSE
  GO TO FIN-PROC
DIVIDE ..................
  .........................
```

Cada vez que el contenido del campo NUMERO es diferente de 55555, el control se cede a la instrucción que sigue al punto, o sea

```
DIVIDE NUMERO BY 2
       GIVING COCIENTE
       REMAINDER RESTO.
```

mientras que, en el caso contrario, se realiza la instrucción alternativa GO TO FIN-PROC.
Debe observarse que a pesar de que las dos formas son completamente equivalentes a los fines del resultado, la segunda es de lectura menos inmediata que la primera. Por tanto, es bueno estructurar los IF de manera sencilla y lineal; esto agilizará tanto la escritura como, y sobre todo, la lectura y manutención del programa.

Condiciones asociables a la instrucción IF. En el formato general de la instrucción IF no se ha especificado el tipo de condiciones que pueden insertarse en las frases condicionales.
Los tests condicionales que pueden actuar en una frase IF son varios, y cada uno de ellos encuentra ocasiones de empleo en muchas fases del proceso. Las diferentes posibilidades, que describiremos en este párrafo, pueden relacionarse someramente así:

- Análisis de relación: permite condicionar la ejecución de una sección del programa a la relación comparativa (mayor, igual, menor, etc.) existente entre dos datos. Este tipo de análisis asume significados particulares cuando se trata de realizar una comparación entre datos alfabéticos o alfanuméricos.

- Análisis de signo: permite condicionar la ejecución en función del signo algebraico de un dato.
- Análisis de condición: permite subordinar la ejecución a la verificación del contenido de un determinado campo.
- Análisis de clase: verifica la clase de pertenencia de un dato (numérico/alfabético) y activa la fase de ejecución adecuada.

Análisis de relación. En el análisis de relación se efectúa una comparación entre dos cantidades (operandos) utilizando un operador relacional especificado.

El formato de la proposición IF para un análisis de relación se indica en la primera tabla de esta página.

El operador califica el tipo de comparación que debe realizarse sobre los operandos especificados, y debe pertenecer al conjunto indicado en la segunda tabla de esta página. Debe observarse que todos los operadores pueden escribirse utilizando tanto el correspondiente símbolo como la propia forma de la lengua inglesa.

Los operandos objeto de la comparación pueden ser campos descritos en DATA DIVISION, constantes o expresiones aritméticas. Por tanto, es posible efectuar análisis comparativos entre operandos que pertenezcan a una cualquiera de las categorías especificadas: un campo puede compararse con otro campo, con una constante o con una expresión aritmética.

Una constante puede compararse con una expresión aritmética, pero no con otra constante.

Cuando uno de los dos operandos interesados en la comparación es una expresión aritmética, antes se calcula el valor de la expresión y, después, este último se coloca en relación con el otro operando. Para aclarar esto supóngase que debe leerse y controlarse un file de fichas cuyo record es

FORMATO DE LA INSTRUCCION IF PARA UN ANALISIS DE RELACION

```
IF  { nombre-de-dato-1 }            { nombre-de-dato-2 }
    { constante-1      }  operador  { constante-2      }
    { expresión-1      }            { expresión-2      }

[THEN]
        { NEXT SENTENCE }
        { frase-1       }

[ ELSE
        { NEXT SENTENCE } ]
        { frase-2       }
```

OPERADORES RELACIONALES

Símbolo Cobol	Símbolo matemático equivalente	Forma completa	Significado
=	=	IS EQUAL [TO]	... igual...
NOT =	≠	IS NOT EQUAL [TO]	... diferente...
>	>	IS GREATER [THAN]	... mayor...
<	<	IS LESS [THAN]	... menor...
NOT >	≤	IS NOT GREATER [THAN]	... menor o igual...
NOT <	≥	IS NOT LESS [THAN]	... mayor o igual...



Finsiel

Sistema IBM 370 utilizado para la gestión de archivos en cinta.

```
01 FICHA-WS.
   05 TIPO-FICHA          PIC X(3).
   05 ARTICULO.
      10 SERIE            PIC X(3).
      10 NUMERO           PIC 9(5).
      10 DESCRIPCION      PIC X(30).
   05 EXISTENCIAS         PIC 9(5).
   05 RESERVA-MINIMA      PIC 9(5).
   05 VENDIDO             PIC 9(5).
   05 MES-VENTAS          PIC 9(2).
```

Supongamos que cada ficha leída deba someterse a los siguientes controles:

1 / Verificar que la ficha tenga TIPO-FICHA igual a ALM; en caso contrario descartarla incrementando un contador de fichas para presentarlo al final del proceso

2 / Verificar que las descripciones del artículo no están vacías; si están vacías debe descartarse la ficha, señalando al final del proceso el total de fichas descartadas

3 / Verificar que la cantidad real (existencias – vendido) no sea inferior a la reserva mínima. En caso contrario, presentar este hecho con un oportuno mensaje

4 / Procesar todos los artículos movidos en el mes tecleado en la consola.

El diagrama de flujo del programa se indica en el gráfico de la página siguiente y el listado puede verse en la pág. 1061.
En el ejemplo se han usado algunos IF para el análisis de relación estre diferentes operandos:

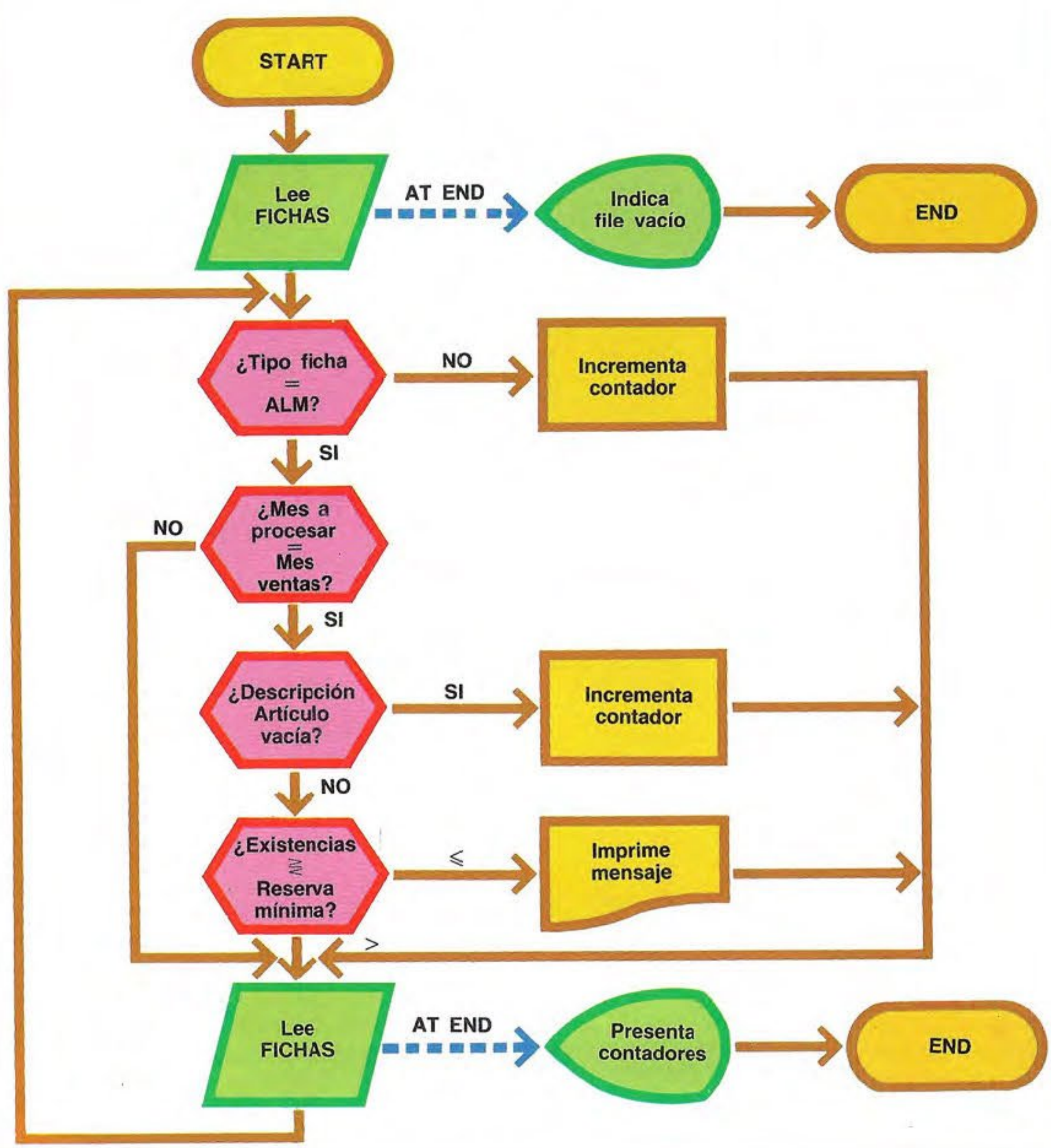
IF TIPO-FICHA NOT = 'ALM'
: compara un campo alfanumérico con una constante alfanumérica

IF MES-VENTAS NOT = MES-PROC
: verifica la igualdad entre dos campos numéricos

IF ARTICULO = SPACES
: relaciona y verifica la igualdad entre un campo alfanumérico y la constante figurativa SPACES; esto equivale a verificar si el contenido de campo son espacios

IF EXISTENCIAS — VENDIDO<RESERVA-MINIMA
: compara el resultado de la expresión aritmética EXISTENCIAS – VENDIDO con el contenido del campo numérico RESERVA-MINIMA

LECTURA Y PROCESO DE UN FILE FICHAS



El análisis de relación puede realizarse tanto entre campos de igual clase (alfanuméricos/numéricos) como entre campos de clase diferente.

Un conocimiento exacto de las reglas de comparación es esencial a los fines del buen funcionamiento de un programa. El programador que no tenga una visión exacta puede incurrir fácilmente en errores que, considerando la amplia gama de comparaciones realizadas, no se evidencian en la compilación. La existencia de un error de este tipo sólo puede evidenciarse con la inexactitud de los resultados o con terminaciones anómalas del proceso a primera vista injustificadas.

Considérese por ejemplo la frase condicional

```
IF SW-ACEPTACION = '0'
   PERFORM PROCESA
ELSE
   PERFORM FIN-PROCESO.
```

LECTURA Y PROCESO DE UN FILE FICHAS

```
01 FICHA-WS.
   05 TIPO-FICHA                PIC X(3).
   05 ARTICULO.
      10 SERIE                  PIC X(3).
      10 NUMERO                 PIC X(5).
      10 DESCRIPCION            PIC X(30).
   05 EXISTENCIAS               PIC 9(5).
   05 RESERVA-MINIMA            PIC 9(3).
   05 VENDIDO                   PIC 9(5).
   05 MES-VENTAS                PIC 9(2).
*
01 MES-PROC                     PIC 9(2).
*
01 CUENTA-LEIDAS                PIC 9(3) COMP VALUE 0.
01 CUENTA-ERRONEAS              PIC 9(3) COMP VALUE 0.
01 CUENTA-DESCARTADAS           PIC 9(3) COMP VALUE 0.

PROCEDURE DIVISION.
INICIO.
   MOVE 0 TO CUENTA-DESCARTADAS
             CUENTA-ERRONEAS
             CUENTA-LEIDAS.
   DISPLAY '** TECLEAR MES A PROCESAR **'
           UPON CONSOLE.
   ACCEPT MES-PROC FROM CONSOLE.
ABRE-FILE.
   OPEN INPUT FICHAS.
PRIMERA-LECTURA.
   READ FICHA INTO FICHA-WS
             AT END
             DISPLAY '** FILE FICHAS VACIO **'
                     UPON CONSOLE
             GO TO FIN-PROC.
CONTROL.
   ADD 1        TO CUENTA-LEIDAS.
   IF TIPO-FICHA NOT = 'ALM'
      ADD 1 TO CUENTA-DESCARTADAS
      GO TO LEE-FICHAS.
   IF MES-VENTAS NOT = MES-PROC
      GO TO LEE-FICHAS.
   IF ARTICULO = SPACES
      ADD 1          TO CUENTA-ERRONEAS
      GO TO LEE-FICHAS.
   IF EXISTENCIAS - VENDIDO LESS THAN RESERVA-MINIMA
      DISPLAY '** ARTICULO : '
             DESCRIPCION
             '    * SERIE : '
             SERIE
             '    * NUMERO : '
             NUMERO
             '    * BAJO-RESERVA **'
             UPON PRINTER.
LEE-FICHAS.
   READ FICHAS INTO FICHA-WS
             AT END
             DISPLAY '***   FICHAS LEIDAS          = '
                     CUENTA-LEIDAS
                     UPON PRINTER
             DISPLAY '***   FICHAS DESCARTADAS     = '
                     CUENTA-DESCARTADAS
                     UPON PRINTER
             DISPLAY '***   FICHAS ERRONEAS        = '
                     CUENTA-ERRONEAS
                     UPON PRINTER
             GO TO FIN-PROC.
   GO TO CONTROL.
*
FIN-PROC.
   CLOSE FICHAS.
   STOP RUN.
```

donde SW-ACEPTACION está descrito en WORKING-STORAGE SECTION del modo

```
01 SW-ACEPTACION        PIC X(3).
```

Si en un cierto momento el proceso SW-ACEPTACION contiene el valor no numérico

0	0	0

el análisis de relación IF SW-ACEPTACION = '0' resulta falso, y el programa realiza la llamada al procedimiento FIN-PROCESO.
En un primer análisis, esto parecería inexplicable, pero el motivo del comportamiento descrito es fácilmente justificable cuando se tiene en cuenta el mecanismo que regula la comparación. La proposición IF permite relacionar campos no numéricos o campos numéricos y no numéricos, cualquiera que sea su longitud. De momento consideraremos campos de igual longitud. En este caso, la comparación se realiza carácter por carácter partiendo de la izquierda, y se interrumpe al terminarse los caracteres a analizar o cuando se encuentra que la condición requerida no se ha satisfecho.
Supóngase por ejemplo los campos CAMPO-1 y CAMPO-2 descritos de la manera

```
01 CAMPO-1          PIC X(3).
01 CAMPO-2          PIC X(3).
```

y supóngase que en un cierto instante éstos valen respectivamente

CAMPO-1 =

A	B	C

CAMPO-2 =

A	B	D

Queremos comprobar la siguiente condición de igualdad:

```
IF CAMPO-1 = CAMPO-2
   DISPLAY 'LOS CAMPOS SON IGUA-
           LES'
           UPON CONSOLE
ELSE
   DISPLAY 'LOS CAMPOS NO SON
           IGUALES'
           UPON CONSOLE
   STOP RUN.
```

El mecanismo de comparación se esquematiza en el gráfico de la página siguiente.

Considérese ahora el caso en que se quiere verificar si el contenido de CAMPO-1 es menor que el contenido de CAMPO-2 para emprender las acciones descritas en la siguiente frase:

```
IF CAMPO-1<CAMPO-2
   DISPLAY 'CAMPO-1 MENOR
           QUE CAMPO-2'
           UPON CONSOLE
ELSE
   DISPLAY 'CAMPO-1 MAYOR O
           IGUAL A CAMPO-2'
           UPON CONSOLE
   STOP RUN.
```

En este caso, el programa asegura ante todo que las longitudes de los dos campos sean iguales y después realiza la comparación (análogamente al ejemplo anterior) carácter por carácter partiendo de la izquierda.
El criterio con que se establece que un carácter tiene valor menor o mayor que otro está ligado a la secuencia ascendente de la representación de los caracteres en el ámbito del código utilizado por la máquina (ASCII o EBCDIC).
En otros términos, considerando los dos valores A B Z y A B 9, resulta ABZ<AB9 cuando el código usado es el EBCDIC, y ABZ>AB9 cuando el código usado es el ASCII.
Efectivamente, según el código EBCDIC, las representaciones de las letras del alfabeto tienen todas valores inferiores a las de los números, mientras que en el código ASCII la situación es opuesta (los códigos de los números tienen valores menores que los de las letras).
Las secuencias de caracteres EBCDIC y ASCII en orden creciente de valor se indican en la tabla de la página siguiente.
Cuando los operandos interesados por una proposición IF no tienen la misma longitud, la comparación se hace como si el operando de longitud inferior estuviese lleno con blanks a la derecha hasta tener la dimensión del operando de mayor longitud.
A este propósito, volvamos a considerar el ejemplo ya indicado

```
IF SW-ACEPTACION = '0'
   PERFORM PROCESA
ELSE
   PERFORM FIN-PROCESO.
```

donde SW-ACEPTACION en el momento del análisis contiene

MECANISMO DE COMPARACION ENTRE DOS CAMPOS ALFANUMERICOS

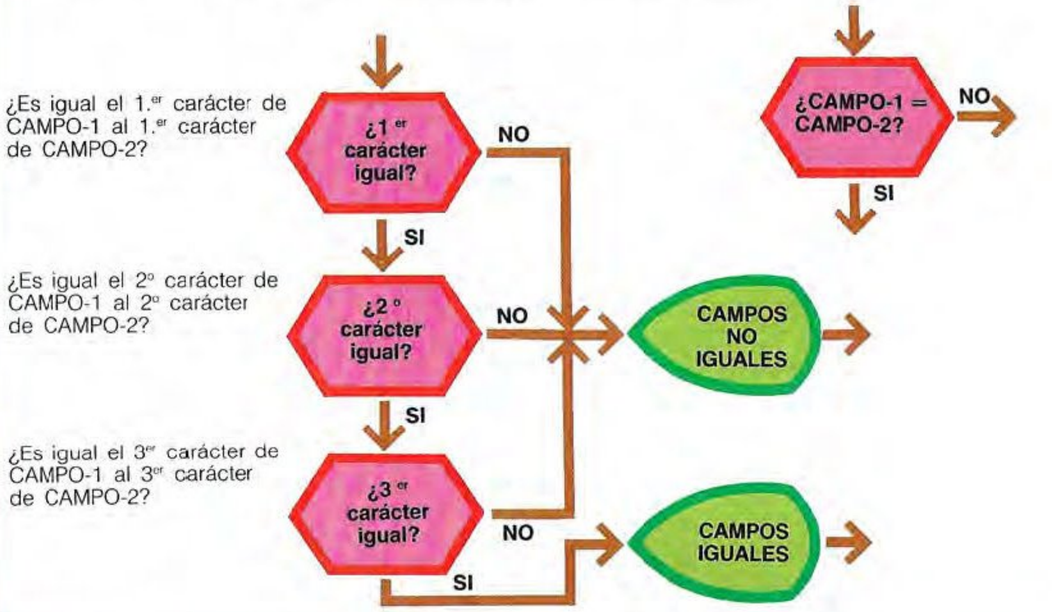


TABLA COMPARATIVA DE LOS CODIGOS EBCDIC Y ASCII

Caracteres en código EBCDIC		Caracteres en código ASCII	
(valores menores)		**(valores menores)**	
ƀ	blank (espacio en blanco)	ƀ	blank (espacio en blanco)
.	punto o punto decimal	"	doble comilla
<	menor	$	dólar
(	paréntesis abierto	'	comilla
+	más	(	paréntesis abierto
$	dólar	)	paréntesis cerrado
*	asterisco	*	asterisco
)	paréntesis cerrado	+	más
;	punto y coma	,	coma
–	menos o trazo de unión	–	menos o trazo de unión
/	barra	.	punto o punto decimal
,	coma	/	barra
>	mayor	0-9	números de 0 a 9
'	comilla	;	punto y coma
=	igual	<	menor
"	doble comilla	=	igual
A-Z	letras de A a Z	>	mayor
0-9	números de 0 a 9	A-Z	letras de A a Z
(valores mayores)		**(valores mayores)**	

0	0	0

Como la constante alfanumérica 0 está constituida por un solo caracter, la comparación se realiza entre los campos

SW-ACEPTACION	0	0	0
Constante '0' extendida a tres caracteres	0		

Como puede observarse, independientemente del código utilizado, los dos valores nunca podrán ser iguales, por lo que el programa llamará siempre el procedimiento FIN-PROCESO.
Contrariamente a lo que sucede para los campos no numéricos, y como es lógico, la comparación entre los campos numéricos no tiene en cuenta las longitudes de los campos comparados, sino sólo su valor algebraico (en la comparación entre campos numéricos se respetan todas las reglas matemáticas), y de ello resulta por ejemplo que

+ 0000005.0000	es igual a 5
− 425	es mayor que − 728
+ 12	es mayor que − 65

Análisis de signo. La proposición IF puede efectuar el análisis del signo del contenido de un campo numérico, o del resultado de una expresión aritmética, utilizando el formato ilustrado en la tabla de abajo. Supóngase que se tienen que calcular las raíces de una ecuación de segundo grado, expresada en la forma

$$Ax^2 + Bx + C = 0$$

Como se sabe, las fórmulas que resuelven la ecuación son

$$x_1 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

$$x_2 = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

Se desean calcular sólo las raíces reales de la ecuación, indicando el caso en que las mismas son complejas conjugadas. Llamando con Δ **(discriminante)** el término B^2-4AC que aparece dentro de la raíz cuadrada, de las matemáticas se sabe que las raíces de la ecuación de 2° grado son:
- reales y diferentes cuando $\Delta > 0$
- reales y coincidentes cuando $\Delta = 0$
- complejas y conjugadas si $\Delta < 0$

Por otra parte, para poder calcular el valor de x_1 y x_2, el valor de A debe ser diferente de 0.
El diagrma de flujo de un programa que calcula las raíces de una ecuación de segundo grado se ha representado en el gráfico de la página siguiente, y el listado en la pág. 1066.
Se recuerda que la raíz cuadrada de un número equivale a elevar la potencia el mismo número con un exponente 1/2 o bien 0.5. Por tanto:

$$\sqrt{B^2 - 4AC} = \sqrt{DELTA} = (DELTA)^{1/2} =$$
$$= (DELTA)^{0.5} = DELTA ** 0.5$$

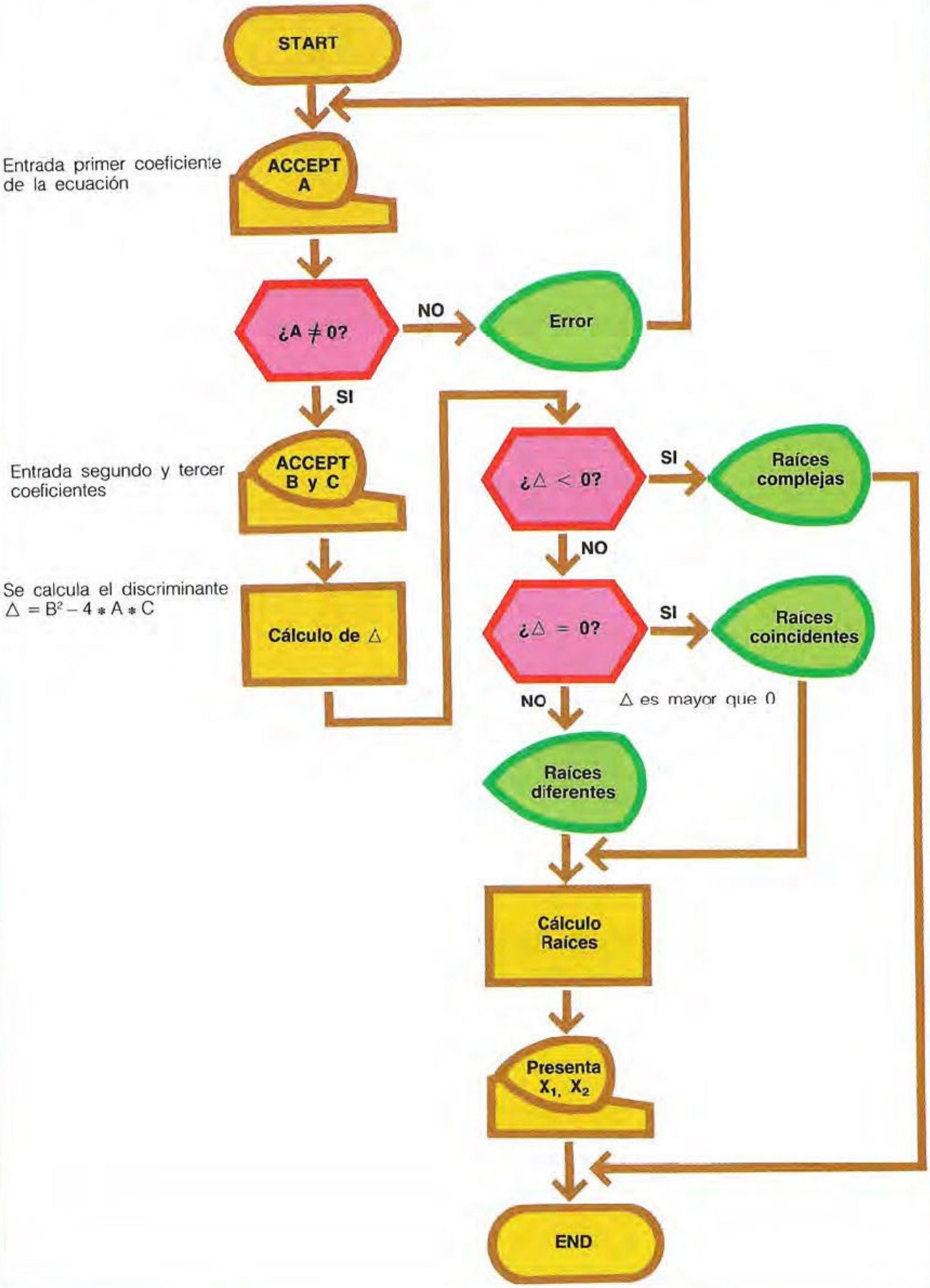
Obsérvese que el cálculo de DELTA se ha efectuado exteriormente a la IF, puesto que era conveniente calcularlo una sola vez en lugar de tres, tantos como eran los análisis de signo previstos. Para concluir podemos añadir que un análisis de signo siempre puede reconducir a un análisis de relación, comparando el valor del campo o de la expresión con 0.

Análisis de condición. Hablando de la descripción de los nombres de los campos de la DATA DIVISION se ha analizado la función del nivel 88. En este contexto se recuerda que si un cam-

FORMATO DE LA INSTRUCCION IF PARA EL ANALISIS DE SIGNO

```
      { nombre-de-dato         }            { POSITIVE }
IF    {                        }  IS [NOT]  { NEGATIVE }
      { expresión-aritmética   }            { ZERO     }
```

CALCULO DE LAS RAICES DE UNA ECUACION DE SEGUNDO GRADO



CALCULO DE LAS RAICES DE UNA ECUACION DE SEGUNDO GRADO

```
 01 A                                     PIC S9(6)V9(3) COMP.
 01 B                                     PIC S9(6)V9(3) COMP.
 01 C                                     PIC S9(6)V9(3) COMP.
 01 DELTA                                 PIC S9(6)V9(3) COMP.
 01 X1                                    PIC S9(6)V9(3) COMP.
 01 X2                                    PIC S9(6)V9(3) COMP.
*
*
 PROCEDURE DIVISION.
 INICIO.
     DISPLAY ' CALCULO ECUACION 2.DO GRADO'
             UPON CONSOLE.
 ACEPTA-A.
     DISPLAY '* TECLEAR PRIMER COEFICIENTE A = '
             UPON CONSOLE.
     ACCEPT A FROM CONSOLE.
     IF A IS EQUAL ZERO
        DISPLAY '*** ERROR ***'
                ' COEFICIENTE A DEBE SER DIFERENTE DE 0'
                UPON CONSOLE
                GO TO ACEPTA-A.
     DISPLAY '* TECLEAR SEGUNDO COEFICIENTE B = '
             UPON CONSOLE.
     ACCEPT B FROM CONSOLE.
     DISPLAY '* TECLEAR TERCER COEFICIENTE C = '
           UPON CONSOLE.
 ACCEPT C FROM CONSOLE.
 COMPUTE DELTA = B ** 2 - 4 * A * C.
 IF  DELTA IS NEGATIVE
     DISPLAY '* DELTA NEGATIVO *
             '* RAICES COMPLEJAS *'
             UPON CONSOLE
     GO TO FIN.
 IF  DELTA IS ZERO
     DISPLAY  '* DELTA NULO *'
              '* RAICES COINCIDENTES *'
              UPON CONSOLE.
 IF  DELTA IS POSITIVE
     DISPLAY '* DELTA POSITIVO *'
             '* RAICES REALES Y DIFERENTES *'
              UPON CONSOLE.
   COMPUTE X1 = (0 - B - DELTA ** .5) / 2 * A.
   COMPUTE X2 = (0 - B + DELTA ** .5) / 2 * A.
   DISPLAY '** X1 = ' X1
           '       '
           '** X2 = ' X2
           UPON CONSOLE.
 FIN.
   DISPLAY 'FIN CALCULO'
           UPON CONSOLE.
   STOP RUN.
```

po se ha descrito con una o más definiciones a nivel 88 no puede analizarse el contenido haciendo referencia directamente a estas definiciones. Los dos ejemplos que siguen evidencian la ventaja que se deriva de utilizar esta posibilidad del Cobol.
Se tiene un record EMPLEADO descrito así:

```
01 EMPLEADO.
   05 MATRICULA        PIC 9(5).
   05 NOMBRE           PIC X(30).
   05 ESTADO-CIVIL     PIC 9.
   88 SOLTERO          VALOR 1.
   88 CASADO           VALOR 2.
   88 VIUDO            VALOR 3.
```

Evidentemente, en el contexto del programa, la instrucción que sigue tiene una interpretación mucho más inmediata.

```
IF VIUDO
   PERFORM LO QUE SEA
```

en lugar de la siguiente

```
IF ESTADO-CIVIL = 3
   PERFORM LO QUE SEA
```

De los ejemplos citados puede verse que el análisis de condición equivale a verificar de forma discursiva el significado lógico del suceso ligado a un cierto contenido del campo.
La ventaja es aún más evidente cuando las condiciones a verificar son muy numerosas y ligadas entre sí en secuencia.
Considérese el caso en que deba leerse un file de fichas en el que los artículos descritos en cada una de ellas tiene un código de dos caracteres; supongamos que, de éstos, el primero deba estar siempre y ser sólo una letra, mientras que el segundo debe estar siempre y ser sólo un número comprendido entre 4 y 9.
Para verificar la validez de cada ficha leída sería necesario verificar todas las posibles combinaciones. Sin embargo, la solución se hace muy sencilla adoptando la siguiente descripción, en la que a SERIE-VALIDA hay asociada toda la secuencia de caracteres que van de A a Z (VALUE 'A' THRU 'Z'), mientras que a NUMERO-VALIDO se asocian los números que van de 4 a 9 (VALUE 4 THRU 9):

```
01 FICHA-ARTICULO
   05 CODIGO-ARTICULO
      10 SERIE-ARTICULO       PIC X.
         88 SERIE-VALIDA      VALUE 'A'
                              THRU 'Z'.
      10 NUMERO-ARTICULO      PIC 9.
         88 NUMERO-VALIDO     VALUE 4
                              THRU 9.
```

Por tanto, la instrucción para el control de CODIGO-ARTICULO puede escribirse:

```
IF NOT SERIE-VALIDA
   PERFORM ERROR-SERIE
   GO TO LEE- FICHAS.
IF NOT NUMERO-VALIDO
   PERFORM ERROR-NUMERO
   GO TO LEE-FICHAS.
.........................................
```

Análisis de clase. El último tipo de análisis que puede realizarse con la instrucción IF es el de clase. Es posible verificar si el contenido de un cierto campo es, en un determinado momento, de clase numérica o alfabética.
El formato de la IF para este tipo de análisis se describe en la primera de las dos tablas de abajo.
Este tipo de análisis requiere el respeto de algunas reglas intuitivas: se sabe que un campo definido alfanumérico (PIC X) puede albergar tanto cifras como letras; para un campo similar tiene sentido efectuar el test para saber si su contenido es NUMERIC o ALPHABETIC; y viceversa, verificar si un campo definido alfabético (PIC A) contiene números es un contrasentido lógico y sintáctico. Es posible reunir en una única tabla (ver tabla al final de la página) los análisis de clase que pueden efectuarse (y no) sobre diver-

FORMATO DE LA INSTRUCCION IF PARA EL ANALISIS DE CLASE

$$\text{IF nombre-de-dato} \quad \text{IS } \underline{\text{[NOT]}} \begin{Bmatrix} \underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}} \end{Bmatrix}$$

ANALISIS DE CLASE

PICTURE	[NOT] NUMERIC	[NOT] ALPHABETIC
A (Alfabético)	NO	SI
9 (Numérico)	SI	NO
X (Alfanumérico)	SI	SI

sos tipos de campo. Además, téngase presente que el campo en que se realiza el control debe ser USAGE DISPLAY, implícita o explícita.

Empleo de los operadores lógicos. Hasta aquí se han descrito todos los análisis que es posible efectuar mediante la proposición IF sobre un solo suceso; es decir, se han tratado condiciones sencillas.
Sin embargo, análogamente a lo que sucede en el pensamiento humano, la acción a emprender depende del resultado de varios análisis interconexos, a menudo de tipo diferente, y efectuados para cualquier número y tipo de variables. La concatenación lógica de varias condiciones puede obtenerse utilizando los operadores lógicos AND, OR, NOT del álgebra booleana.

Siempre que una decisión esté subordinada a la verificación simultánea de dos o más sucesos, el análisis puede conducirse con un solo IF utilizando el operador lógico AND.

Esto hace que la frase condicional resulte verdadera siempre que la primera condición y la segunda resulten verdaderas.
Considérese por ejemplo

```
01 CAMPO-1        PIC 9 VALUE 1.
01 CAMPO-2        PIC 9 VALUE 0.
```

Si durante la ejecución quiere verificarse si el contenido de los dos campos todavía es el implantado inicialmente, puede escribirse

```
IF      CAMPO-1 = 1
        AND
        CAMPO-2 = 0
        PERFORM CAMPOS-OK
ELSE    ..................................
```

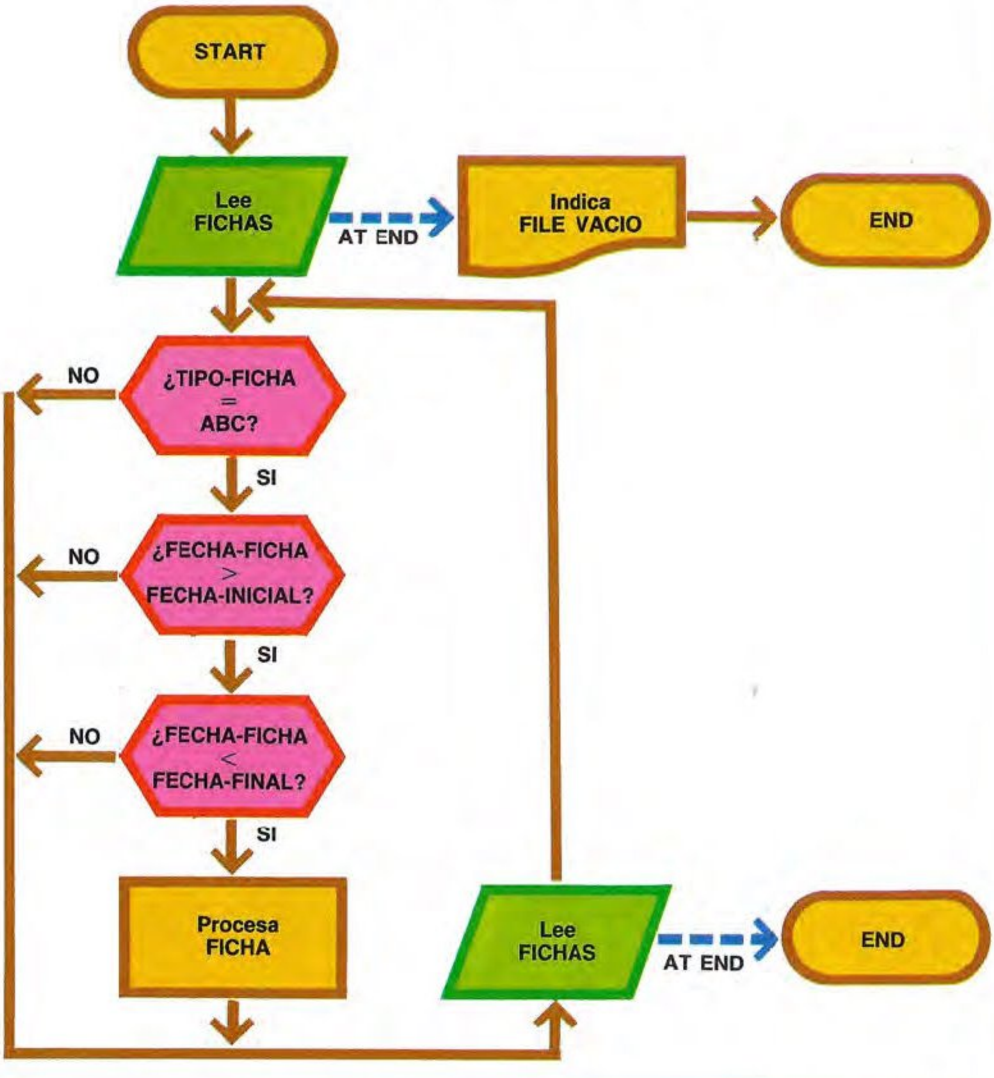
La llamada al procedimiento CAMPOS-OK se activa sólo cuando los dos campos cumplan simultáneamente las condiciones requeridas: es suficiente que una de las dos no se verifique para que el control pase a las instrucciones subordinadas a la cláusula ELSE.
Un ejemplo muy frecuente de aplicación del operador AND es el relativo a la elección de los records de entrada de un file sobre la base de la pertenencia o no de un dato a un intervalo precisado.
Considérese el caso en que las fichas del file FICHAS tengan la siguiente descripción:

```
01 FICHA-WS.
   05 TIPO-FICHA              PIC X(3).
         88 FICHA-VALIDA      VALOR 'ABC'.
   05 FECHA-FICHA.
      10 AÑO-FICHA            PIC 9(2).
      10 MES-FICHA            PIC 9(2).
      10 DIA-FICHA            PIC 9(2).
   05 .................................   ..............
```

EJEMPLO DE USO DEL OPERADOR AND

```
PROCEDURE DIVISION.
INICIO.
    OPEN INPUT FICHAS.
PRIMERA-LECTURA.
    READ FICHAS INTO FICHA-WS
                AT END
                DISPLAY '** FILE FICHAS VACIO **'
                        UPON PRINTER
                CLOSE FICHAS
                STOP RUN.
CONTROL.
    IF FICHA-VALIDA
       AND
       FECHA-FICHA > FECHA-INICIAL
       AND
       FECHA-FICHA < FECHA-FINAL
       PERFORM PROCESA-FICHA.
    READ FICHAS INTO FECHA-WS
                AT END
                CLOSE FICHAS.
                STOP RUN.
    GO TO CONTROL.
*
*
*
```

EJEMPLO DE USO DEL OPERADOR AND



Supongamos que sólo deben procesarse las fichas de tipo 'ABC' y correspondientes al período comprendido (extremos excluidos) entre

```
01 FECHA-INICIAL.
   05 AÑO-INIC          PIC 99.
   05 MES-INIC          PIC 99.
   05 DIA-INIC          PIC 99.


01 FECHA-FINAL.
   05 AÑO-FIN           PIC 99.
   05 MES-FIN           PIC 99.
   05 DIA-FIN           PIC 99.
```

La fase de elección y proceso de las fichas puede detallarse como se ve en el listado de la página anterior, que corresponde al diagrama de flujo representado arriba.
Es suficiente que la ficha no sea del tipo ABC, o que uno de los dos límites del intervalo no sea respetado, para que el control pase a la instrucción siguiente que cierra la IF. Sin embargo, por esta instrucción se pasa cualquiera que sea el resultado de la IF, puesto que la READ no está subordinada a la cláusula ELSE. La validez y la claridad de la solución propuesta pueden apreciarse observando, por comparación, el siguiente detalle del mismo párrafo CONTROL:

```
CONTROL.
    IF FICHA-VALIDA
       IF FECHA-FICHA>FECHA-INICIAL
          IF FECHA-FICHA<FECHA-FINAL
             PERFORM PROCESA-FICHA.
```

En este caso se ha utilizado una serie de instrucciones IF anidadas.

Adoptar este tipo de codificación, sobre todo en los casos menos sencillos que el descrito, comporta generalmente una notable pérdida de legibilidad del programa y un gran incremento de las posibilidades de error.

Una serie de IF anidadas conduce al lector a niveles siempre más profundos, haciéndole perder el hilo lógico del análisis realizado por el programa.

Si para emprender una determinada acción es suficiente que se verifique una sola de una serie de condiciones, es posible expresar la frase condicional recurriendo al operador lógico OR.

El siguiente ejemplo es una aplicación de los operadores lógicos AND y OR usados para censar los empleados de una empresa no casados y con fecha de nacimiento comprendida entre el 31 de diciembre de 1940 y el 1 de enero de 1960 (extremos incluidos). El file que contiene los datos es el file de fichas NOMINA. El diagrama de flujo correspondiente al ejemplo se ha representado en la página siguiente, y el listado del programa en la página 1072.

El significado del operador NOT es intuitivo, pero es necesario poner sobre aviso al lector respecto a su utilización, puesto que a menudo el programador, al codificar una frase condicional en que se usa el NOT, altera el sentido real de dicha frase.

Esta posibilidad es prácticamente imposible en el caso de IF sencillas, en las que debe evaluarse la verificación o no de un solo suceso; la siguiente frase

```
IF A NOT = B
   PERFORM PRIMERA-ACCION
ELSE
   PERFORM SEGUNDA-ACCION.
```

no presenta ninguna duda, puesto que el sentido de la instrucción es de lectura inmediata.

No resulta tan inmediata la comprensión de una frase con una o más condiciones negadas conectadas con otros operadores lógicos AND y OR, puesto que éstas respetan las reglas generales del álgebra booleana. Sin entrar a fondo en el tema, considérese a continuación el siguiente ejemplo.

Debe leerse el file EMPLEADOS, en el que cada ficha contiene el campo relativo al estado civil del empleado.

```
01 EMPLEADO.
   05 .................
   05 .................
   05 ESTADO-CIVIL          PIC 9.
      88 SOLTERO            VALOR 1.
      88 CASADO             VALOR 2.
      88 VIUDO              VALOR 3.
   05 .................
      .................
```

Durante la lectura del file de fichas se quiere controlar la validez del campo ESTADO-CIVIL: si éste contiene un número igual a 1, 2 o 3, el proceso puede continuar; en caso contrario, es necesario emitir una oportuna señal de error.

En codificación Cobol puede escribirse como sigue:

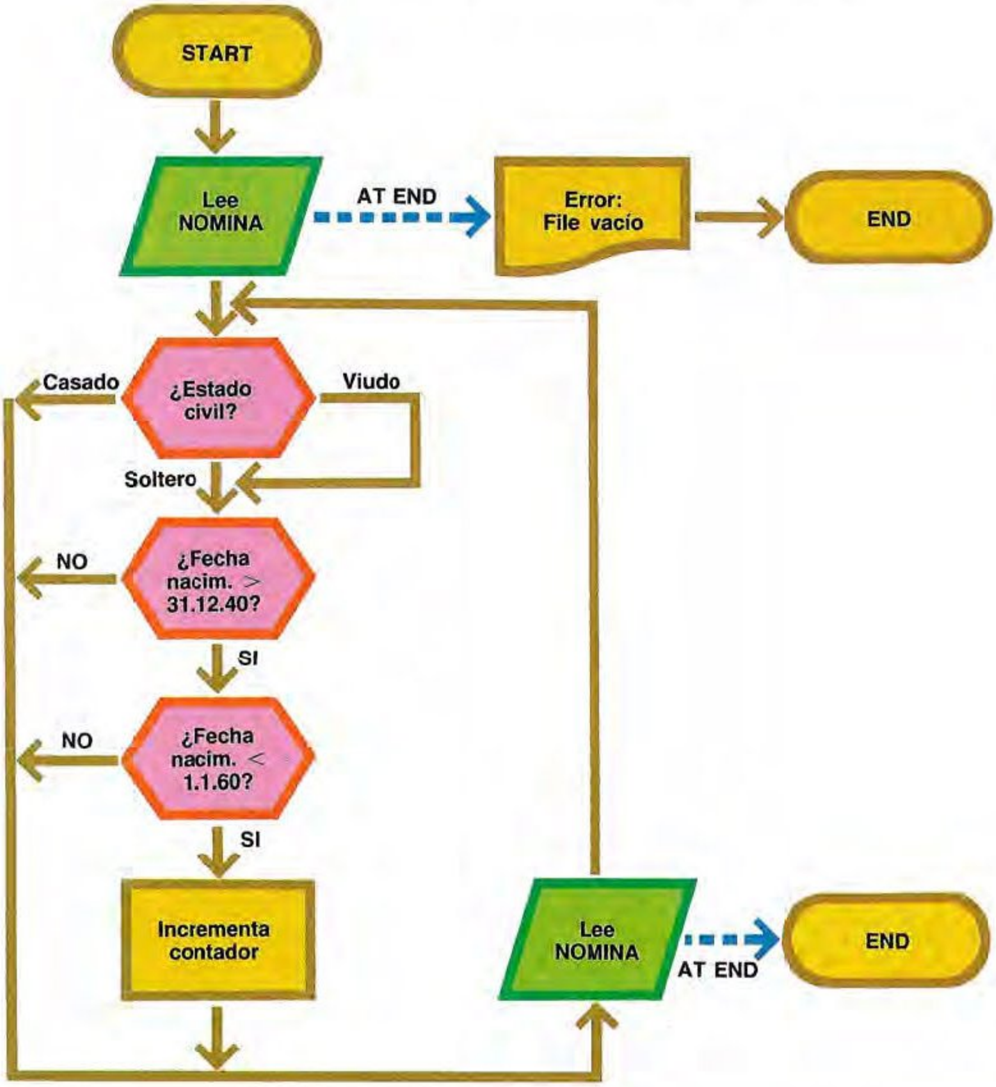
```
IF SOLTERO
   OR
   CASADO
   OR
   VIUDO
   PERFORM PROCESA
ELSE
   DISPLAY 'ESTADO CIVIL ERRONEO'
           UPON CONSOLE.
```

Si por motivos de estructuración del programa debe analizarse el caso de error, o bien controlar si el estado civil no es soltero, ni casado, ni viudo, puede pensarse intuitivamente en codificar las instrucciones de la forma siguiente:

```
IF NOT SOLTERO
      OR
   NOT CASADO
      OR
   NOT VIUDO

   DISPLAY 'ESTADO CIVIL ERRONEO'
           UPON CONSOLE
ELSE
   PERFORM PROCESA.
```

EJEMPLO DE USO DE LOS OPERADORES AND Y OR



En realidad, adoptando las instrucciones descritas, todas las fichas se señalarán como si fuesen erróneas.
Efectivamente, supóngase que el empleado descrito en la ficha a procesar sea soltero. En este caso, la primera condición para descartar la ficha no se verifica, ya que es verdad que el empleado es soltero; sin embargo, siguiendo en el análisis de la frase, el empleado (soltero) no puede ser casado y, por tanto, se verifica la condición de error.
El análisis de la frase continúa hasta la verificación para viudo, pero en este punto ya se han verificado dos condiciones de error, por lo que se emite la indicación 'ESTADO CIVIL ERRONEO', aunque en realidad, el significado de este mensaje no es exacto.
La explicación de lo dicho se expresa sintéticamente con la siguiente fórmula:

NOT (A OR B) = (NOT A) AND (NOT B)

La negación global de una serie de condiciones alternativas (OR) puede expresarse **sólo** como producto lógico (AND) de las negaciones de las condiciones componentes.

EJEMPLO DE USO DE LOS OPERADORES AND Y OR

```
 01 EMPLEADO-WS.
     05 MATRICULA                     PIC 9(3).
     05 NOMBRE                        PIC X(30).
     05 ESTADO-CIVIL                  PIC 9.
         88 SOLTERO                   VALUE 1.
         88 CASADO                    VALUE 2.
         88 VIUDO                     VALUE 3.
     05 FECHA-NACIMIENTO.                 .
         10 AÑO-NACIMIENTO            PIC 9(2).
         10 MES-NACIMIENTO            PIC 9(2).
         10 DIA-NACIMIENTO            PIC 9(2).
      05 .............
      05 .............
          10 ..........
*
*
 01 CONTADOR                          PIC 9(4) COMP VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 INICIO.
     OPEN INPUT NOMBRE.
 PRIMERA-LECTURA.
     READ NOMBRE INTO EMPLEADO-WS
                 AT END
                 DISPLAY '** FILE NOMBRE VACIO **'
                              UPON PRINTER
                 CLOSE NOMBRE
                 STOP RUN.
 CONTROL.
     IF (SOLTERO OR VIUDO)
          AND
          FECHA-NACIMIENTO > 401231
          AND
          FECHA-NACIMIENTO < 600101
          ADD 1 TO CONTADOR.
     READ NOMBRE INTO EMPLEADO-WS
                      AT END
                      CLOSE NOMBRE
                      DISPLAY '** EMPLEADOS NO CASADOS '
                                   'PEDIDOS = '
                                   CONTADOR
                                   UPON PRINTER.
                      STOP RUN.
*
*
```

Por tanto, la codificación correcta de la segunda forma de la frase debe ir como la que se ha escrito a continuación:

```
IF NOT SOLTERO
        AND
 NOT CASADO
        AND
  NOT VIUDO
        DISPLAY, ESTADO CIVIL ERRONEO'
                   UPON CONSOLE
ELSE
  PERFORM PROCESA.
```

Para finalizar, debe subrayarse que en caso de que se deban analizar más condiciones simultáneamente, es una buena norma controlar la secuencia de las condiciones reuniendo los grupos sencillos entre paréntesis. De esta manera se tiene la doble ventaja de evitar errores y de hacer de fácil interpretación el sentido lógico del análisis efectuado por el programa.

La instrucción GO TO

Todas las instrucciones de un programa Cobol se realizan en la misma secuencia en que las ha escrito el programador, a menos que éste no

altere el orden de ejecución recurriendo a la instrucción de salto GO TO.
Un programa Cobol podría estar constituido en teoría por una serie de instrucciones todas pertenecientes a un párrafo único como se indica a continuación:

```
PROCEDURE DIVISION.
INICIO.
    OPEN ..........
    READ ..... INTO .....
    ADD ..........
    MOVE ..........
    WRITE ..........
    ......................
    STOP RUN.
```

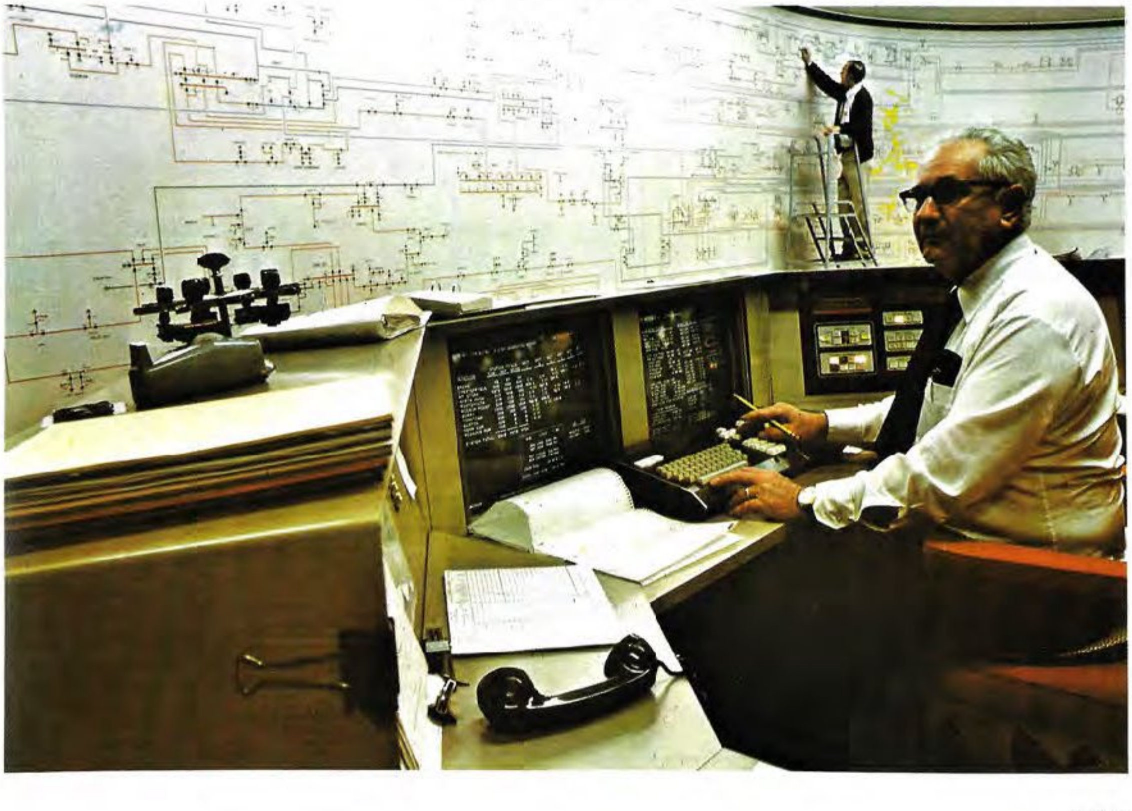
Es del todo evidente que un programa de este tipo sólo puede realizar funciones extremadamente sencillas. La repetición de una o más operaciones siempre iguales, como por ejemplo la lectura de un file y el proceso de sus records, impone alterar el flujo estrechamente secuencial del programa creando fases iterativas sobre un cierto número de instrucciones oportunamente reunidas.

Sin embargo, se sabe que un programa Cobol puede estar organizado en párrafos y secciones. Esta organización es particularmente funcional cuando, asociada al uso del verbo PERFORM, permite crear puntos de enganche a los cuales puede cederse el control utilizando la instrucción GO TO. Para el Compilador, los nombres de párrafos y de secciones son completamente transparentes, a menos que no se llamen con una instrucción GO TO. El formato más sencillo con que puede escribirse esta instrucción es:

GO TO nombre-de-procedimiento

donde nombre-de-procedimiento es un nombre compuesto como máximo por 30 caracteres alfanuméricos perforados a partir del margen A (columna 8).
Un salto de un punto a otro del programa puede estar condicionado a la verificación de un suceso previsible o incondicionado y, por tanto, introducido por el programador para pasar al procedimiento indicado.
Las dos situaciones descritas se evidencian en los siguientes ejemplos:

Cuadro de control de una central eléctrica de distribución.



K. Reese/Marka

Primer caso

```
ADD 1 TO CONTADOR.
IF CONTADOR > 100
   GO TO FIN
ELSE
   .....
   .....
```

Segundo caso

```
MOVE A TO B.
ADD 100 TO K.
GO TO PROCEDIMIENTO-1.
```

En el primer caso, el salto al procedimiento FIN se efectúa sólo cuando el contenido del CONTADOR supera el valor 100, mientras que en el segundo caso, el salto a PROCEDIMIENTO-1 se realiza en cualquier caso después de haber sumado 100 a la variable K. El programador es libre de usar las instrucciones de salto en cualquier punto del programa, pero debe tenerse en cuenta que el abuso de este tipo de instrucción conduce inevitablemente a un producto de escasa legibilidad. Y viceversa, un uso escaso y limitado de GO TO hace más evidente el flujo lógico de las operaciones, con gran ventaja para el mantenimiento, la depuración y la claridad del programa.

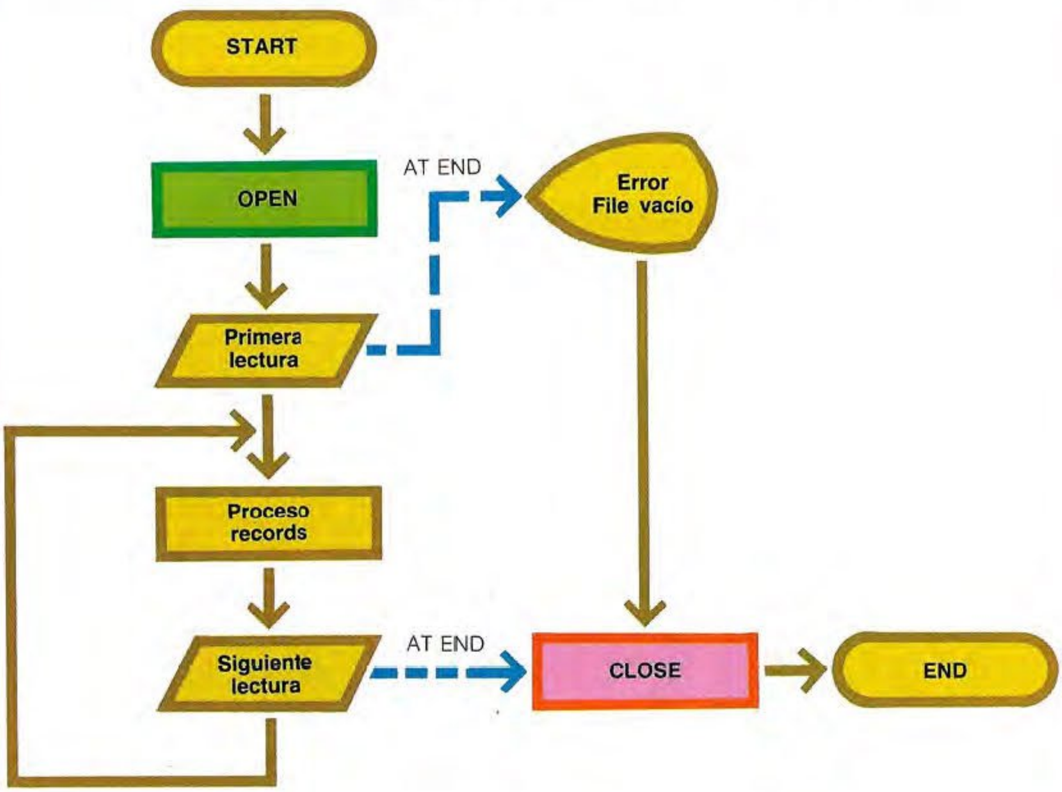
Un ejemplo típico de utilización de GO TO es el que se tiene en la lectura de un file y en el control de las condiciones de fin de file. En muchos de los ejemplos indicados, puede observarse que el flujo de las operaciones de lectura de un file secuencial respeta el esquema de la figura de abajo.

La primera lectura, llamada también «lectura fuera de ciclo», se realiza una sola vez, inmediatamente después de la apertura del file.

Es evidente que la verificación de las condiciones de fin de file en este contexto denuncia que el file está vacío. Naturalmente, una condición similar es anómala y, por tanto, se jusifica el cierre del proceso.

Si el file no está vacío, el control pasa a la primera instrucción a realizar después de la READ, y el record leído se procesa.

CICLO DE LECTURA DE UN FILE SECUENCIAL



CICLO DE LECTURA DE UN FILE SECUENCIAL

```
.................
     OPEN INPUT FILE-IN.
 PRIMERA-LECTURA.
     READ FILE-IN INTO REC-IN
                    AT END
                    DISPLAY '** FILE FILE-IN VACIO **'
                                UPON CONSOLE
                    GO TO FIN-PROCESO.
 PROCESA-RECORD.
     MOVE ........... TO ..........
     ADD  ........... TO ..........
                         ..........
                         ..........
     IF   ........... = ...........
          ..........
          COMPUTE ...... = .......... + ..........
          MOVE    ...... TO ..........
     ELSE .........
          .........
          .........
     MOVE REC-IN      TO ..........
*
 LEE-FILE-IN.
     READ FILE-IN INTO REC-IN
                AT END
                DISPLAY '** FIN LECTURA FILE-IN **'
                            UPON CONSOLE
                GO TO FIN PROCESO.
     GO TO PROCESA-RECORD.
*
 FIN-PROCESO.
     CLOSE FILE-IN.
     STOP RUN.
```

Terminada esta fase se tiene una segunda instrucción de lectura, que se realizará tantas veces como records haya en el file.
El detallado del ciclo completo resulta tener la estructura indicada en el listado de arriba.

La cláusula DEPENDING ON. GO TO se utiliza generalmente después de la verificación de la producción de un cierto suceso: según el resultado de la indagación efectuada, el control se direcciona hacia los oportunos procedimientos. El caso de empleo más difundido se describe en el siguiente ejemplo:

```
IF A = 1
  ...........
  ...........
  GO TO A-IGUAL-1
ELSE
  ...........
  GO TO A-DIFERENTE-1.
```

De este modo es posible ceder el control a los dos procedimientos A-IGUAL-1, A-DIFERENTE-1 según el resultado del test.

En el caso en que las condiciones a verificar sean más de una, puede resultar muy enojoso escribir

```
IF A = 1
  GO TO PROCEDIMIENTO-1.
IF A = 2
  GO TO PROCEDIMIENTO-2.
IF A = 3
  GO TO PROCEDIMIENTO-3.
  ...........
  ...........
IF A = N
  GO TO PROCEDIMIENTO-N.
```

La cláusula DEPENDING ON ligada a la instrucción GO TO permite obtener el mismo resultado de esta manera:

```
GO TO PROCEDIMIENTO-1
      PROCEDIMIENTO-2
      PROCEDIMIENTO-3.
      ........................
      PROCEDIMIENTO-N DEPENDING
      ON A.
```

El flujo de operaciones que el Compilador activa en base a dicha codificación es exactamente igual al descrito anteriormente.
Sin embargo, debe precisarse que

- el campo especificado en la cláusula DEPENDING ON (A en el ejemplo) debe ser un campo numérico
- si el contenido del campo especificado es un número diferente de 1, 2, ..., N (los procedimientos relacionados), el control pasa a la instrucción que sigue al comando de salto
- los nombres de los procedimientos deben relacionarse de manera que el número contenido en el campo especificado en la cláusula DEPENDING ON identifique su posición en la relación.

Por ejemplo, la frase

```
GO TO A
      B
      C
      D
      E DEPENDING ON NUMERO.
```

pasará el control al procedimiento A si NUMERO = 1, al procedimiento B si NUMERO = 2, y así sucesivamente.

El verbo PERFORM

La instrucción PERFORM, ya introducida y utilizada en algunos de los ejemplos presentados, permite enviar a ejecución con un único comando todas las instrucciones pertenecientes a un párrafo o a una sección. Esta posibilidad permite subdividir el programa en bloques funcionales que pueden reclamarse en cualquier punto del flujo del proceso.
El empleo de esta instrucción, junto al carácter descriptivo propio del Cobol, permite estructurar el programa de forma sintética y autodocumentada.
Naturalmente, esta última característica está subordinada a que el programador use, para cada nombre de procedimiento invocado por el verbo PERFORM, nombres explicativos de la función que realiza dicho procedimiento.
Si, por ejemplo, una sección de programa efectúa el cálculo del ITE es aconsejable llamarla CALCULA-ITE y no con siglas de tipo C-I o de tipo similar.
La instrucción PERFORM está dotada de muchas cláusulas, que se analizarán a continuación; el formato base al que se hará referencia en las notas de introducción es el siguiente:

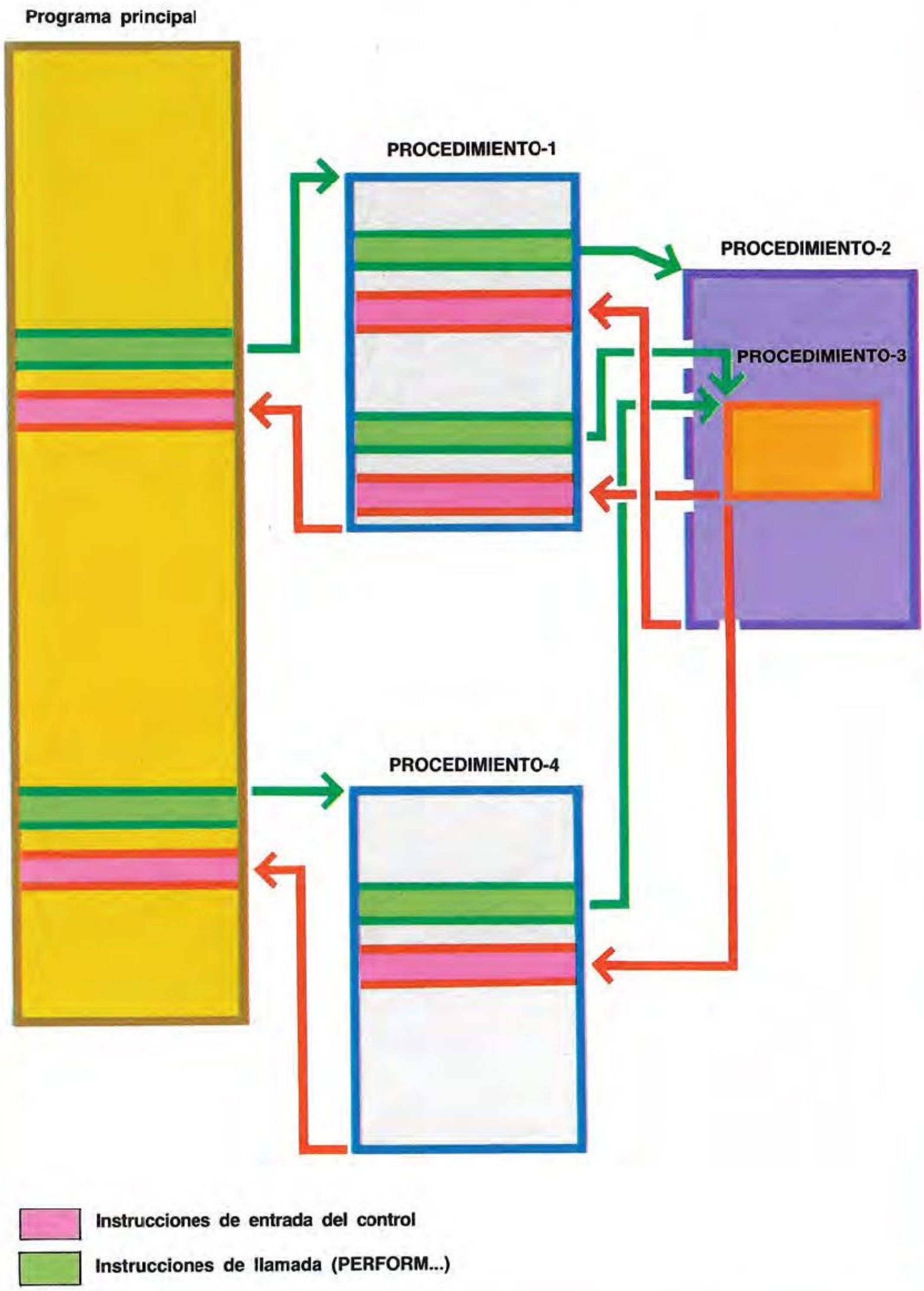
PERFORM nombre-de-procedimiento

donde «nombre-de-procedimiento» es el nombre de un párrafo o de una SECTION.
En la página siguiente se ha esquematizado la dinámica mediante la cual el programa realiza todas las instrucciones contenidas en un procedimiento cuando éste es invocado en una PERFORM. En la figura también es posible ver cómo el control de las operaciones pasa del programa principal a los procedimientos llamados y cómo de éstos se restituye a la primera instrucción que debe realizarse inmediatamente después del verbo de llamada.
Por otra parte es posible observar que un procedimiento puede contener la llamada a otro procedimiento, tanto si éste es externo como si está contenido en el que llama.
Además, un procedimiento puede llamar un número cualquiera de veces desde un punto cualquiera del programa.
En el esquema citado aparece, entre los diversos procedimientos invocados, también un PROCEDIMIENTO-3 que resulta ser una subparte del PROCEDIMIENTO-2. Por sencillez, supóngase que las funciones de PROCEDIMIENTO-3 sean asumidas por el conjunto de instrucciones relacionadas abajo y comprendidas entre el nombre de párrafo PARRAFO-1 y el nombre de párrafo PARRAFO-4:

```
PARRAFO-1.
       MOVE A TO B.
       ...........  ┐
       ...........  │
PARRAFO-2.          │
       ...........  │  PROCEDIMIENTO-3
       ...........  │
PARRAFO-3.          │
       ...........  │
       ...........  ┘
       ADD 1 TO K.
PARRAFO-4.
       ...........
       ...........
```

Es posible llamar con una sola instrucción todas las indicadas con el corchete escribiendo sencillamente

LLAMADAS DE PROCEDIMIENTO MEDIANTE PERFORM



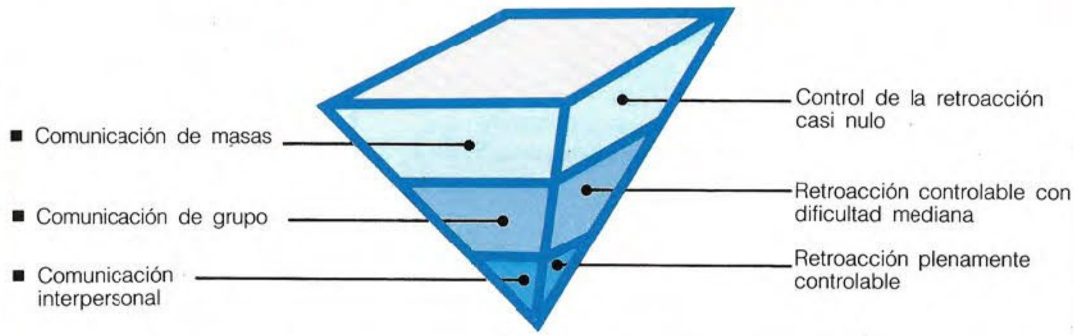
El diálogo del hombre con las máquinas

Tomando al azar una jornada típica puede analizarse, desde la mañana a la noche, cuántas veces se ha tenido ocasión de:

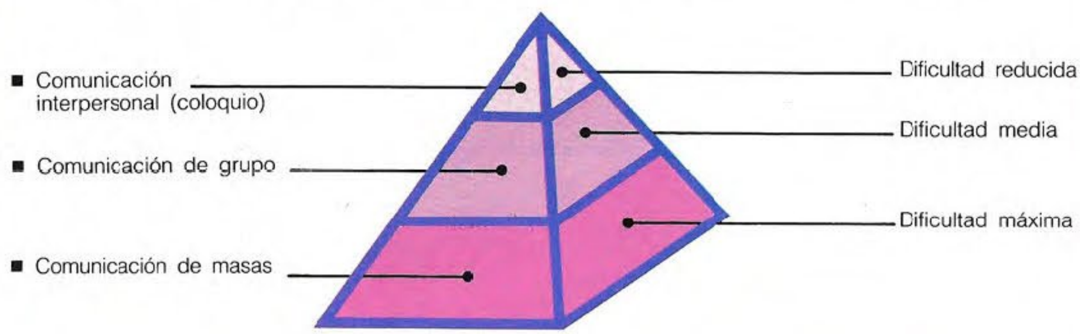
- *hablar con un solo interlocutor* (comunicación interpersonal).
- *hablar a un grupo de personas* (comunicación de grupo).
- *hablar frente a un medio de masas, p.e. radio, TV, etc.* (comunicación de masas).

Evidentemente, a menos que no se sea un gran presidente, la mayor parte de las ocasiones serán de tipo interpersonal.
Esto indica un orden de frecuencia de uso de las comunicaciones netamente a favor de los coloquios entre personas sencillas, lo que comporta un entrenamiento notable para este tipo de comunicaciones, y muy poco entrenamiento para las comunicaciones de grupo o de masa.
La consecuencia es el temor a hablar en público y el pánico de la grabación (audio o vídeo), que nos hace aparecer peores comunicadores de lo que en realidad somos.
La dificultad de comunicación podría representarse en una escala de valores del tipo indicado en la segunda parte de la figura de abajo.
¿Por qué todo esto?
Entre muchos motivos, uno es el principal: la gradual reducción del control sobre las comunicaciones de retorno (retroacción o feedback) que sigue un proceso igual a la escala de dificultades indicada antes, pero invertido (ver la primera parte de dicha figura).
El fenómeno puede explicarse mediante lo que podría definirse como la teoría del murciélago. *Como se sabe, el murciélago es un animal que ve poco, pero posee por don natural una especie de sistema de radar que puede emitir señales que, reflejadas por los obstáculos situados en su camino, le revelan su existencia y le permite evitarlos. Es un sistema de análisis de las señales de retorno.*
En el coloquio, el hombre se comporta de la misma manera. Observa a su interlocutor y ob-

CONTROL DE LA RETROACCION ASOCIADO A TRES TIPOS DE COMUNICACION



GRADOS DE DIFICULTAD ASOCIADOS A TRES TIPOS DE COMUNICACION



tiene una serie de señales: el movimiento de los ojos, de la boca, de las manos son códigos precisos que indican atención, enojo, prisa, estupor, etc. Basta con pensar en la señal de atención que envía inconscientemente el que escucha con los ojos fijos, la cabeza quieta, o sea completamente dedicado a escuchar, en contraposición con el que escucha tamborileando los dedos sobre la mesa, mirando a menudo el reloj, bostezando, etc.
Este tipo de control de las señales de retorno también puede producirse en la comunicación de grupo, aunque el grado de dificultad aumenta a causa del número de interlocutores a tener bajo control simultáneamente.
En cambio, desaparece completamente cuando nuestra comunicación es directa a un micrófono, o a una cámara de TV o de cine. Por esto, para muchas personas, hablar ante un ojo de vidrio inexpresivo y minucioso en su silencio pasivo, resulta aterrorizador. Lo mismo puede suceder frente a un teclado de ordenador.
Esto se debe a que la ausencia de señales humanas de retorno los inhibe. Pero con una observación más atenta, se verá que entre estas señales buscamos (equivocándonos a veces) recoger sólo «refuerzos» positivos.
Es decir, buscamos ayuda, *incluso sin darnos cuenta. Efectivamente, una de las funciones fundamentales de la comunicación es la del* continuo alivio de la ansiedad, *junto a muchas otras. El hombre, además de* placer *tiene* necesidad *de comunicar. Richard Stevens relaciona ocho funciones de base alrededor de las cuales se desarrolla la actividad de relación humana. Comunicamos*

- *para realizar o conseguir cualquier cosa* (Funciones instrumentales)
- *para hacer que cualquiera se comporte de una determinada manera* (Función de control)
- *para descubrir o explicar cualquier cosa* (Función informativa)
- *por el interés que se encuentra en una situación dada* (Función de estimulación)
- *porque la situación lo requiere* (Función ligada al papel)
- *para expresar nuestros sentimientos o para imponernos en alguna forma* (Función expresiva)
- *por el gusto de estar en compañía* (Función de contacto social)
- *para airear un problema, dar salida a una preocupación* (Función de alivio de la ansiedad).

Como puede observarse, al menos tres de las funciones descritas, la expresiva, *la de* contacto social *y la de* alivio de la ansiedad, *evidencian la necesidad de intercambiar relaciones humanas con los demás. Veamos cómo Stevens ilustra más ampliamente estas tres funciones.*

Función expresiva. *Una mujer se confía a una amiga. Un hombre reprende con ira el error de un colega. Un enamorado susurra tiernas palabras de amor. La comunicación es un proceso que permite expresar nuestro modo de sentir: puede ser espontánea y auténtica, o bien cuidadosamente construida para conseguir un fin deseado. Esta deliberada presentación puede producirse para conseguir objetivos instrumentales: por ejemplo, para causar buena impresión a un potencial suministrador de trabajo durante un coloquio. La función autoexpresiva de la comunicación puede actuar de maneras disimuladas y sutilmente evasivas. Lo que durante una recepción podría parecer sólo un intercambio casual de informaciones, en realidad puede ser un intento de manifestar agudeza e inteligencia; una discusión de política puede constituir para uno de los interlocutores la ocasión de manifestar la propia superioridad sobre otro.*
El estilo de la comunicación, el lenguaje empleado, las palabras que se utilizan, pueden constituir la manera de afirmar la pertenencia a un grupo. Muchas subculturas, grupos profesionales o de otro tipo tienen una jerga específica, que debe conocerse y usarse en la forma apropiada para poder ser miembros de ellas.

Funciones de contacto social. *La comunicación puede ser un fin en sí misma. Como en todas las características humanas, también en este caso hay diferencias de individuo a individuo. Sin embargo, la mayor parte de nosotros manifiesta en mayor o menor medida el deseo de la compañía de otras personas por el simple gusto de estar juntas. Buscamos de forma particular el contacto con personas de nivel y experiencia similares a los nuestros. Los experimentos psicológicos, por un lado, y los relatos de náufragos o de eremitas por otro, demuestran que el aislamiento social suele producir resultados bastante devastadores.*

Función de alivio de la ansiedad. *Está ampliamente documentado el hecho de que cuando un individuo se encuentra en una situación que induce ansiedad, las más de las veces tiende a buscar el contacto con los demás. El psicólogo americano Stanley Schachter ha encontrado, por ejemplo, que algunas estudiantes que creían que debían someterse a tests psicológicos en los que se les suministrarían dolorosas descargas eléctricas preferían en gran medida (con respecto a otro grupo de chicas a las que se les había dicho que las descargas no serían dolorosas) esperar su turno junto con otras. Esto sucedía sólo si las compañeras que esperaban se encontraban en la misma condición. Schachter (1959) resumía así la sistuación: «la infelicidad no se contenta con una compañía cualquiera, sino que tiene necesidad de la compañía de otros infelices».*

Por tanto es claro que la comunicación es útil para intercambiar una compleja serie de mensajes, muchos de ellos ricos en implicaciones secundarias. No hace mucho, Levi Strauss definía la sociedad como un sistema de individuos y de grupos humanos que comunican entre sí. Por tanto, la comunicación no sólo es un sistema de lenguajes, sino un proceso de personas.

También se ha visto cuánto sirve la comunicación para los contactos humanos, indispensables para aquellas relaciones de ayuda *necesarias al individuo para sobrevivir sin demasiadas angustias.*

Finalmente también se ha visto cómo estas relaciones, tanto para la costumbre de los intercambios personales que privilegian los contactos entre personas sencillas, como por los contactos mediante mass media, ponen de relieve el diálogo *entre individuos.*

¿Pero qué sucede cuando el diálogo no puede establecerse entre dos personas porque se trata de un hombre y de una máquina?

Nace un nuevo tipo de diálogo. Cuando la máquina es un ordenador, nace un diálogo especial, hecho de humanidad *por una de las dos partes y de* rapidez de ejecución *por otra.*

Por tanto, todo transcurre bien cuando el hombre, frente al ordenador, pide soluciones rápidas basadas en los programas que él mismo u otros hombres en su lugar han proyectado.

Sin embargo, las cosas empiezan a no funcionar cuando el hombre no busca, como sería lo correcto, sólo rapidez, sino que también pide otro tipo de ayuda. En sustancia, cuando busca el diálogo, *entendido como lo hemos definido antes, o sea un proceso de intercambio de relaciones, de sentimientos, de fuga de la sociedad o del miedo.*

Es ilegítimo pedir al ordenador ayuda en el sentido psicológico, pero a menudo, se hace inconscientemente. De ahí las preguntas ansiosas de muchos padres. «¿No será nocivo para el desarrollo intelectual del niño?» «¿No lo hará dependiente de la máquina?» «¿No será una ayuda aparente, mientras que en realidad podría producirse un daño en su capacidad de desarrollo del pensamiento?»

Ante todo, cada generación debería tener el buen sentido de no interferir excesivamente en las necesidades de la siguiente: es decir, no debería intentar regirla con el propio patrón de metro mental por una sencilla razón: la siguiente generación NO TIENE las mismas necesidades que la anterior.

Asumamos entonces un primer hecho: HA CAMBIADO EL TIPO DE RELACION EN LOS DIALOGOS. La cantidad de diálogos entre personas cada vez irá disminuyendo, con toda la ventaja para los diálogos entre el hombre y el ordenador. Quien pretenda obtener del ordenador la misma función comunicativa que obtenía en el diálogo humano, vive fuera del tiempo. Cada hombre debe entrenarse para dialogar con la máquina y no sentirse frustrado por ello.

Piénsese cómo ha cambiado la relación con el teléfono, que es uno de los medios más difundidos de comunicación. Hoy, con el teléfono se hace todo; parece que sin él casi no se podría vivir. Y sin embargo, sólo hace pocas decenas de años era considerado una especie de monstruo negro del que había que desconfiar.

Actualmente, muchas organizaciones de asistencia telefónica salvan vidas humanas, e incluso alivian la ansiedad a los deprimidos, gracias a los contactos telefónicos. Sin embargo, se dirá que más allá del hilo hay una presencia humana y que, por tanto, aunque mediada, la presencia del hombre no desaparece totalmente.

Es cierto: el software con el que hay que dialogar o con el que se juega al ajedrez no es, en términos psicológicos lo mismo. Pero en esta relación ¿es la máquina la que debe modificarse o es el hombre? Si pedimos a la máquina calor humano, sentimientos, ayuda psicológica y no los obtenemos ¿es que la máquina es defectuosa? No.

No es legítimo pedir lo que no puede tenerse.

Por tanto, el ordenador termina por convertirse en un test de equilibrio psicológico. Si no nos sentimos bien con la máquina, es que no nos sentimos bien con nosotros mismos. Quizá tenemos necesidad de relaciones de ayuda, pero éste es un problema que no corresponde al ordenador.
El ordenador es un instrumento. Los intrumentos, por definición, son neutros. El cuchillo sirve para cortar el pan o para degollar a un adversario: esto no depende del cuchillo, sino del uso que se hace del mismo. Por tanto, podría decirse que una persona acostumbrada a tener un buen equilibrio consigo misma (no obligada a depender de las ayudas comunicativas ajenas), podría «dialogar» con el ordenador sin ningún problema.
Los psicoanalistas freudianos saben muy bien que la «neutralidad« del analista la vive el paciente, en las primeras sesiones, como una forma de hostilidad. «No colabora... Está allí para oírme hablar... Pero ¿me toma el pelo...?».
Esto indica hasta qué punto *exigimos de nuestros interlocutores una intervención* activa. *La neutralidad de la persona con la que intentamos dialogar (aunque el psicoanálisis no es un diálogo) nos irrita, hasta que comprendemos que la*


SIP



J. Launois/Grazia Neri-Black Star

La dificultad de comunicación reduce a menudo la capacidad y el interés de los estudiantes. La didáctica gestionada por el ordenador (a la derecha) derriba la barrera de la relación alumno-maestro. En estas condiciones, el control de la retroacción es total porque la máquina es un medio neutro. Soluciones innovadoras como la teleconferencia (arriba) introducen un interfaz electrónico entre las personas que se comunican, aunque hacen siempre posible dicha comunicación.

respuesta a nuestros problemas no está en el otro, *sino en nosotros mismos.*

Si no sabemos «descubrirnos», no será ni un diván ni un teclado lo que nos hará daño, aunque será cómodo transferir nuestra angustia a estos instrumentos pasivos, acusándolos de su pasividad.

Sin embargo, en el caso de un individuo que se acerque al uso del ordenador con un correcto y equilibrado enfoque psicológico, el tipo de lenguaje utilizado reviste una gran importancia.

Como se ha visto, no es correcto pedir una ayuda en términos psicológicos al ordenador, mientras que es legítimo pedir que, después de haber pretendido durante mucho tiempo que el hombre se adaptase a la máquina, las máquinas se adapten al hombre no sólo en términos ergonométricos, sino en términos de expresividad.

Por expresividad no sólo debe entenderse una mejora lingüística *en los diálogos hombre-máquina, por otra parte ampliamente mejorables, porque la comunicación entendida como* palabra escrita *es sólo una de las formas expresivas tanto del hombre como del ordenador.*

La expresividad del ordenador se entiende como una forma de comunicación directa al hombre que, aunque neutra, no es irritante. Los colores y los sonidos, además de las palabras, son un universo todavía por explorar en el nuevo enfoque hombre-máquina. Pero el escaso entrenamiento para entender la comunicación también como un hecho visible y sonoro a menudo conduce a dar una mayor importancia al aspecto lingüístico, a la palabra escrita.

El advenimiento de la escritura informática traerá, y en ciertos aspectos ya la ha traído, una revolución parecida a la aportada por Gutenberg con la invención de los caracteres móviles de impresión. La comunicación escrita deberá ser más concreta, más sintética. Será necesario aprender a transmitir emotividad y expresividad sin recurrir necesariamente a ríos de términos triunfalistas. ¿Cómo? El camino ya está trazado por los técnicos de las comunicaciones escritas, los primeros los periodistas y los redactores de textos publicitarios. He aquí algunas sugerencias prácticas para hacer más adaptada al medio electrónico la comunicación escrita:

- Usar palabras de pocas sílabas. *Según Rudolph Flesh, un párrafo de 100 palabras escritas en inglés debería tener 200 sílabas*
- Usar frases cortas, *o sea compuestas de*
- *sujeto, verbo y predicado con una media de unas 15 palabras por frase*
- Dar informaciones útiles a quien recibe el mensaje. *Es necesario preguntarse constantemente «¿qué recibe el lector de esta afirmación?»*
- No intentar decir demasiadas cosas a la vez. *Quien quiere decir demasiado no dice en realidad nada*
- Explicar a través de ejemplos. *Oportunamente elegidos, los ejemplos aclaran mucho más que muchas definiciones detalladas*
- Anteponer el ejemplo a la definición. *Si tuviese que explicarse el efecto sinérgico, podría empezarse con un ejemplo: «dos más dos igual a cinco. El sinergismo...». En cambio, a menudo se hace lo contrario en la comunicación escrita: primero se define el tema y después se pone un ejemplo. Así sucede que tiene que volverse a leer todo desde el principio, puesto que la definición sólo se entiende después de haber leído el ejemplo*
- Anticipar las conclusiones, *según el esquema conocido como narración invertida típica del lenguaje periodístico: primero la conclusión del hecho y después la cronología. También aquí se suele hacer lo contrario, narrando todo lo ocurrido sin decir la conclusión hasta el final. Es una técnica que puede dejarse para los folletines*
- Repetir. *Aumenta la memorización. Para evitar la creación de una reacción negativa («pero esto ya lo he leído...») debe* repetirse *de otra manera que la utilizada anteriormente. Las formas de repetición más útiles son el* caso *indicado como situación de referencia, que lo repite todo en forma práctica tomada de la realidad, y el* resumen final, *que siempre es conveniente hacer para esquematizar todo lo que se ha dicho anteriormente.*

¿Vamos a resumir también nosotros?

El ordenador es un instrumento y, por tanto, un medio neutro. No le pidamos una ayuda psicológica como estamos acostumbrados a hacer en los diálogos con las personas, porque el diálogo hombre-máquina tiene otra finalidad. Aceptemos, pues, la nueva realidad: menos comunicaciones interpersonales como fuga de la angustia y más comunicación orientada al intercambio de informaciones.

Enrico Cogno

```
PERFORM          PARRAFO-1
THRU             PARRAFO-3
```

En el formato indicado, la PERFORM confía el control a la primera instrucción de PARRAFO-1 (MOVE A TO B) y lo recupera rápidamente después de la ejecución de la última instrucción de PARRAFO-3 (ADD 1 TO K.)

Es interesante subrayar que el uso de este formato del verbo PERFORM presenta dos inconvenientes fundamentales que inducen a desaconsejar su empleo.

El primer inconveniente viene dado por la posibilidad de incurrir en el error de escribir

```
PERFORM          PARRAFO-1
THRU             PARRAFO-4
```

tratando de enviar a ejecución sólo las instrucciones comprendidas entre las palabras PARRAFO-1 y PARRAFO-4.

En realidad, la instrucción escrita envía a ejecución también todas las instrucciones de PARRAFO-4, lo que conduce a resultados totalmente imprevisibles. Ahora bien, este obstáculo puede superarse con la experiencia.

En cambio, el segundo inconveniente está ligado a la mantenibilidad del programa y, por tanto, tiene un peso mayor que el primero. Efectivamente, supóngase que, para una actualización, debe modificarse el programa de la siguiente manera:

```
PARRAFO-1.
    MOVE A TO B
    ...........
    ...........
PARRAFO-2.
    ...........
    ...........
PARRAFO-3.
    ...........
    ...........
REENTRADA.
    ...........
    ...........
    IF A NOT = C
        ...........
        ...........
        GO TO REENTRADA.
    ...........
    ...........
    ADD 1 TO K.
PARRAFO-4.
    ...........
    ...........
```

El ordenador personal Hewlett-Packard HP87.



R. Vila/Marka

Se ha creado el nuevo párrafo REENTRADA entre PARRAFO-3 y PARRAFO-4, cuya presencia está motivada por la necesidad de efectuar un control sobre la variable A.
En este caso, la instrucción

```
PERFORM   PARRAFO-1
THRU      PARRAFO-3
```

restituye el control al programa principal apenas encuentra el párrafo REENTRADA, alterando el sentido del proceso.
Por tanto es necesario corregir todas las instrucciones de llamada modificándolas así:

```
PERFORM PARRAFO-1 THRU REENTRADA.
```

La entidad de la corrección necesaria puede ser notable, y depende del número de modificaciones hechas y del volumen del programa.
En conclusión, es desaconsejable estructurar el programa en SECTION que agotan completamente las funciones a las que van destinadas, y efectuar las llamadas siempre en la forma

```
PERFORM nombre-de-sección.
```

Si se desea adoptar la llamada de más procedimientos mediante la proposición THRU, puede recurrirse a la instrucción EXIT.

La instrucción EXIT. La instrucción EXIT no tiene en realidad ningún efecto.
Gracias a esta característica, es posible obviar en parte el segundo inconveniente indicado al hablar del verbo PERFORM. Es suficiente que el programa esté estructurado así:

```
PARRAFO-1.
    MOVE A TO B
    ...........
    ...........
PARRAFO-2.
    ...........
    ...........
PARRAFO-3.
    ...........
    ...........
  ADD 1 TO K
PARRAFO-3-EX.
    EXIT
PARRAFO-4.
```

En este caso, la instrucción

```
PERFORM   PARRAFO-1
THRU      PARRAFO-3-EX.
```

envía a ejecución, como última instrucción, la EXIT que no tiene efecto, puesto que el control se restituye nuevamente a la parte que llama. De esta manera, un párrafo cualquiera insertado en el interior del procedimiento llamado se ejecuta sin interrupciones.

Ejecuciones iterativas del mismo procedimiento. La instrucción PERFORM permite realizar el mismo procedimiento, o el mismo grupo de procedimientos (THRU), un número prefijado de veces utilizando la cláusula TIMES.
El formato de la PERFORM completo de las cláusulas descritas hasta ahora se ha representado en la página siguiente.
Por ejemplo, supóngase que debe efectuarse la suma de los números enteros de 1 a 1000.
El programa de càlculo puede tener la siguiente forma:

```
PROCEDURE DIVISION.
INICIO SECTION.
PON A CERO.
     MOVE ZEROES TO NUMERO
                    TOTAL.
INCREMENTA-NUMERO.
     ADD 1 TO NUMERO.
     IF NUMERO GREATER 1000
       DISPLAY '*** TOTAL ='
               TOTAL
               '***'
               UPON CONSOLE
      STOP RUN.
     ADD NUMERO TO TOTAL.
     GO TO INCREMENTA-NUMERO.
*
```

Con las siguientes instrucciones puede obtenerse el mismo resultado:

```
PROCEDURE DIVISION.
INICIO SECTION.
PON A CERO.
   MOVE ZEROES TO NUMERO
                  TOTAL.
     PERFORM SUMA 1000 TIMES.
     DISPLAY '*** TOTAL ='
             TOTAL
             '***'
             UPON CONSOLE.
```

PRIMER FORMATO DE LA INSTRUCCION PERFORM

```
PERFORM  nombre-de-procedimiento-1
         [THRU  nombre-de-procedimiento-2]

           ⎡⎧ número-entero-1 ⎫        ⎤
           ⎢⎨                 ⎬  TIMES ⎥
           ⎣⎩ nombre-de-dato  ⎭        ⎦
```

```
     STOP RUN.
*
 SUMA SECTION.
 SUMA-PARA.
     ADD 1 TO NUMERO.
     ADD NUMERO TO TOTAL.
     SUMA-EX. EXIT.
```

La cláusula UNTIL. La posibilidad de conocer a priori el número de veces que debe ejecutarse un grupo de instrucciones es bastante rara, incluso si el ciclo de iteración utiliza un campo numérico implantado por el programa, y no una constante fija como en el último ejemplo considerado.
Lo más probable es que desee bloquearse la iteración en el momento en que se verifica un cierto suceso.
Por ejemplo, queremos sumar los primeros números enteros hasta que su suma no sea mayor de 3425.
Debe tenerse en cuenta que el valor de detención de la iteración también puede estar contenido en un campo numérico definido en WORKING-STORAGE.
El resultado deseado se obtiene con las siguientes instrucciones, que comprenden el uso de la cláusula UNTIL (hasta que):

```
PROCEDURE DIVISION.
INICIO SECTION
PON A CERO.
   MOVE ZEROES TO NUMERO
                  TOTAL
                  NUMEROS-SUMADOS.
   PERFORM SUMA
           UNTIL
           TOTAL GREATER 3425.
   DISPLAY '*** NUMEROS-SUMADOS ='
           NUMEROS-SUMADOS
           UPON CONSOLE.
   DISPLAY '*** TOTAL FINAL ='
           TOTAL
           UPON CONSOLE.
   STOP RUN.
*
 SUMA SECTION
 SUMA-PARRAFO.
     ADD 1 TO NUMERO
              NUMEROS-SUMADOS.
     ADD NUMERO TO TOTAL.
 SUMA-EX. EXIT.
```

La cláusula VARYING... FROM... BY. Esta cláusula extiende las potencialidades de la anterior, puesto que permite condicionar las iteraciones a la verificación de una condición (UNTIL), pero está dotada de una mayor flexibilidad. Esta flexibilidad se debe a la posibilidad de variar (VARYING), durante la ejecución del procedimiento, el valor de un dato cualquiera.
Este valor se incrementa con cada iteración un determinado paso (BY), partiendo de un valor inicial deseado (FROM).
El formato de la instrucción PERFORM que incluye también la cláusula examinada se ha representado en la tabla de la pág. 1086.
Una aplicación típica de este formato corresponde a la gestión de las tablas, sobre todo en las fases de carga y de búsqueda secuencial. De este aspecto se hablará ampliamente más adelante.

La instrucción STOP. El formato de esta instrucción, ya conocida de los ejemplos presentados es el siguiente:

```
STOP RUN
```

Sin embargo, existe un segundo formato que permite detener el proceso sólo temporalmente,

SEGUNDO FORMATO DE LA INSTRUCCION PERFORM

```
PERFORM   nombre-de-procedimiento-1
          [THRU nombre-de-procedimiento-2]
          VARYING    nombre-de-dato-1
                     { constante-numérica-1 }
          FROM       {                      }
                     { nombre-de-dato-2     }
                     { constante-numérica-2 }
          BY         {                      }
                     { nombre-de-dato-3     }
          UNTIL      condición
```

enviando simultáneamente un mensaje a la consola del sistema.
El final de la pausa lo decreta la respuesta del operador y el proceso se reemprende, en base a la respuesta facilitada, desde la primera instrucción que sigue a la STOP.
La línea

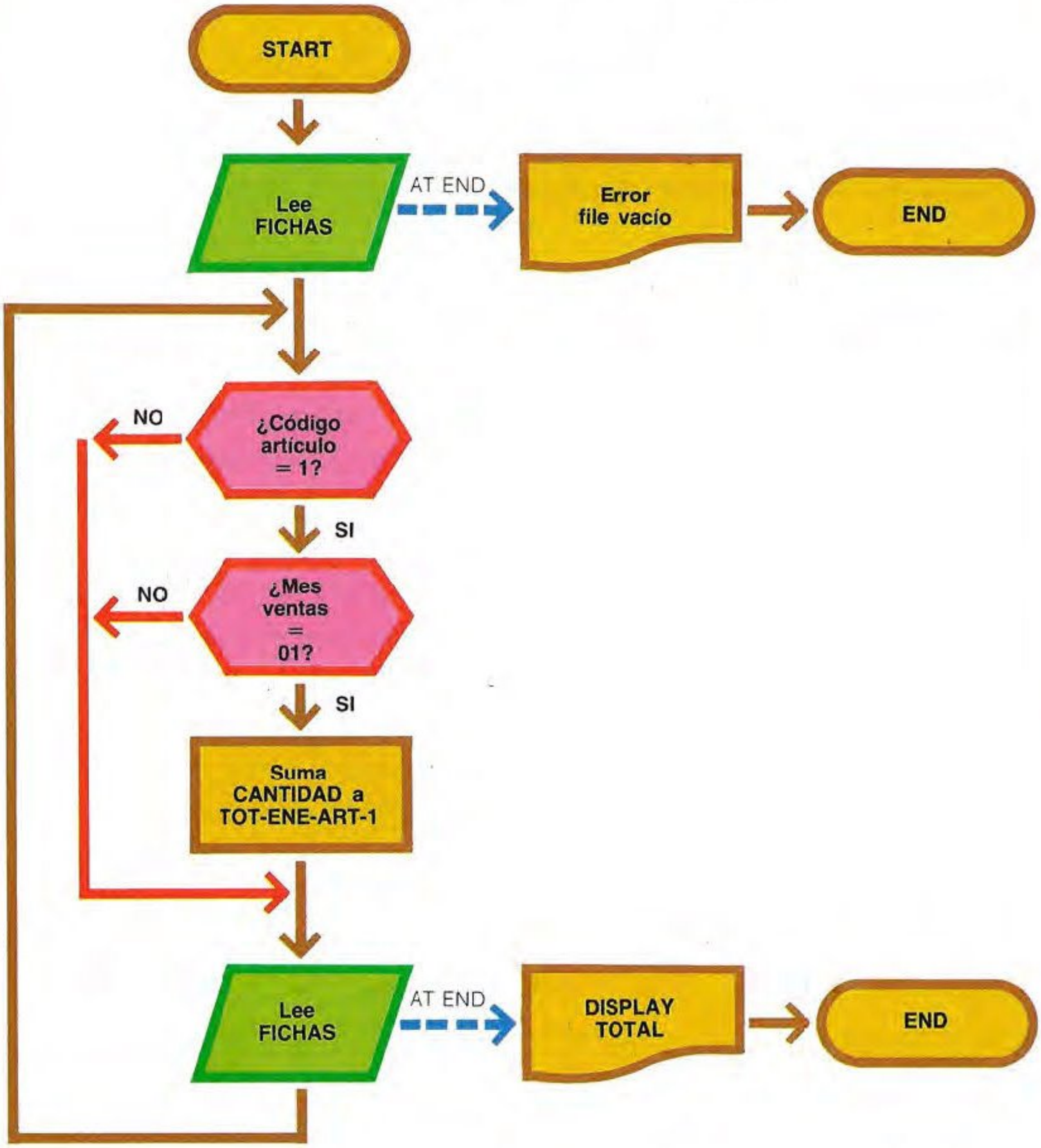
STOP 'ERROR DE CALCULO'

le envía a la consola el mensaje ERROR DE CALCULO.

Gestión de las tablas en el Cobol

En el Cobol, una tabla está constituida por un conjunto de campos contiguos en la memoria destinados a contener informaciones homogéneas. Las tablas constituyen uno de los instrumentos más potentes de que dispone el Cobol para la gestión de los datos en la memoria. Efectivamente, si para pequeñas cantidades de datos (a gestionar directamente en la memoria), el recurso a esta técnica puede ser una elección del programador, es prácticamente imposible hacerlo cuando los datos a manipular son muy numerosos.
Considérese el caso en que debe leerse un file de fichas en el que cada ficha indica la cantidad de artículos vendidos en un almacén. Al final de la lectura de todo el file debe imprimirse el total de las cantidades vendidas en el mes de enero solamente del artículo de código 1.
Evidentemente, en este caso basta con un programa muy sencillo, cuyo diagrama de flujo se ha representado en la página siguiente.
Observando la figura puede verse cómo, después de haberse asegurado de que la ficha leída se refiere al artículo de código 1 y a las ventas de enero, se calcula la suma de la cantidad vendida en el único totalizador definido: TOT-ENE-ART-1. El programa puede detallarse como se ve en el listado de la pág. 1088.
Para tener un esquema visible que será útil más adelante, imagínese una memoria reservada a los datos como una hoja, en el que el totalizador TOT-ENE-ART-1 ocupa la parte indicada en la figura de arriba de la pág. 1089.
Consideremos ahora el caso en que se quiere calcular la suma de las cantidades vendidas, siempre del artículo de código 1, también para todos los demás meses del año.
En este caso, el diagrama de flujo del programa se complica notablemente, incluso siendo todavía limitado el número de las informaciones a gestionar, como puede deducirse del examen de la figura de abajo de la pág. 1089.
La traducción del diagrama de flujo en lenguaje Cobol da lugar al programa del listado de las págs. 1090 y 1091.
La utilización de totalizadores separados, como se ha hecho ahora, es incómodo y poco elegante cuando el número de totalizadores se hace superior a 3 o 4 y, además, es imposible cuando este número suele ser del orden de centenares o de millares.

LECTURA Y PROCESO DE UN FILE (1)



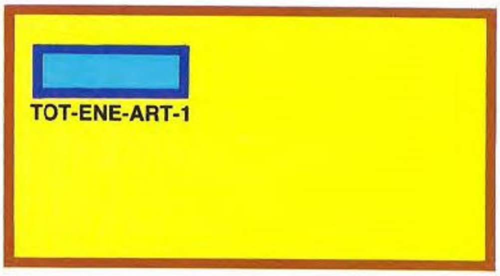
FLUJO DE LOS DATOS



LECTURA Y PROCESO DE UN FILE (1)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABLAS.
REMARKS. ESTE PROGRAMA ES UN EJEMPLO
         DE INTRODUCCION AL USO DE LAS
         TABLAS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. ...........
OBJECT-COMPUTER. ...........
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FICHAS ASSIGN TO CARD-READER.
*
*
DATA DIVISION.
FILE SECTION.
FD FICHAS
   LABEL RECORD OMITTED.
01 FICHA                          PIC X(80).
*
*
*
WORKING-STORAGE SECTION.
01 FICHA-WS.
   05 SERIE-ARTICULO              PIC 9.
   05 CODIGO-ARTICULO             PIC 9.
   05 FECHA-VENTA.
      10 AÑO                      PIC 99.
      10 MES                      PIC 99.
      10 DIA                      PIC 99.
   05 CANTIDAD                    PIC 9(3).
   05 COSTO-UNITARIO              PIC 9(3).
   05 FILLER                      PIC X(66).
*
*
01 TOT-ENE-ART-1                  PIC 9(6) COMP VALUE 0.
*
*
PROCEDURE DIVISION.
INICIO.
   OPEN INPUT FICHAS.
PRIMERA LECTURA.
   READ FICHA INTO FICHA-WS
              AT END
              DISPLAY '*** FILE FICHAS VACIO ***'
                      UPON PRINTER
              GO TO FIN-PROCESO.
CONTROL.
   IF CODIGO-ARTICULO NOT = 1
      GO TO LEE-FICHAS.
   IF MES = 1
      ADD CANTIDAD TO TOT-ENE-ART-1.
*
*
LEE FICHAS.
   READ FICHAS INTO FICHA-WS
              AT END
              DISPLAY ' TOTAL VENTAS ART. 1 ENERO =
                      TOT-ENE-ART-1
                      UPON PRINTER
              GO TO FIN-PROCESO.
   GO TO CONTROL.
*
*
FIN-PROCESO.
   CLOSE FICHAS.
   STOP RUN.
```

ESQUEMA DE LA MEMORIA CON UN SOLO TOTALIZADOR

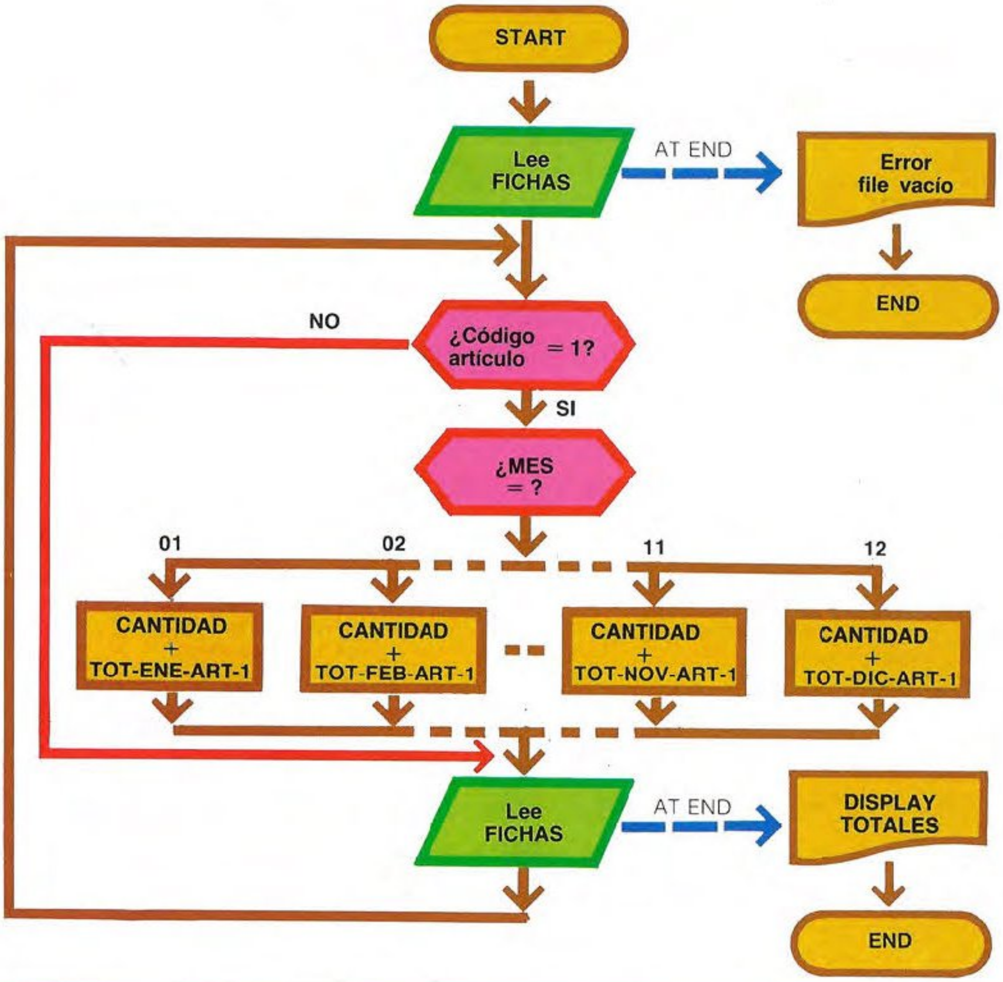


Volviendo a considerar la memoria disponible para los datos como una hoja, la parte ocupada por los contadores reservados a las cantidades del artículo de código 1 vendidas cada mes puede esquematizarse como se indica en la figura de arriba de la pág. 1092.

Tablas de una dimensión

Los problema indicados tienen una solución muy sencilla. Los totalizadores pueden declararse al Compilador como formando parte de una tabla de 12 elementos, y es posible direccionar cada uno de ellos con un nombre genérico de elemento y con el número que identifica su posición en el ámbito de la tabla.
Es decir, se crea una tabla de totales mensuales

LECTURA Y PROCESO DE UN FILE (2)

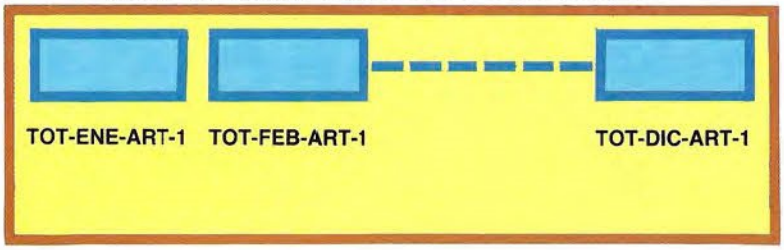


LECTURA Y PROCESO DE UN FILE (2)

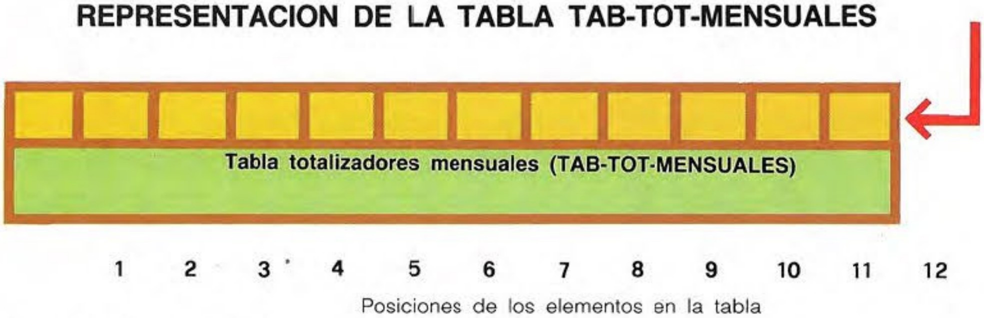
```
 WORKING-STORAGE SECTION.
 01 FICHA-WS.
     05 SERIE-ARTICULO             PIC 9.
     05 CODIGO-ARTICULO            PIC 9.
     05 FECHA-VENTA.
         10 AÑO                    PIC 9(2).
         10 MES                    PIC 9(2).
         10 DIA                    PIC 9(2).
     05 CANTIDAD                   PIC 9(3).
     05 COSTO-UNITARIO             PIC 9(3).
     05 FILLER                     PIC X(66).
*
*
* - - - - - - - - - - - - - TOTALIZADORES - - - - - -   - -
*
*
*
 01  TOT-ENE-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-FEB-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-MAR-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-ABR-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-MAY-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-JUN-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-JUL-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-AGO-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-SEP-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-OCT-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-NOV-ART-1                 PIC 9(6)  COMP  VALUE 0.
 01  TOT-DIC-ART-1                 PIC 9(6)  COMP  VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 INICIO.
     OPEN INPUT FICHAS.
 PRIMERA-LECTURA.
     READ FICHAS INTO FICHA-WS
                     AT END
                     DISPLAY '** FILE FICHAS VACIO **'
                             UPON PRINTER
                     GO TO FIN PROCESO.
 CONTROL.
     IF CODIGO-ARTICULO = 1
         NEXT SENTENCE
     ELSE
         GO TO LEE-FICHAS.
     GO TO SUMA-ENE
           SUMA-FEB
           SUMA-MAR
           SUMA-ABR
           SUMA-MAY
           SUMA-JUN
           SUMA-JUL
           SUMA-AGO
           SUMA-SEP
           SUMA-OCT
           SUMA-NOV
           SUMA-DIC DEPENDING ON MES.
*
*
*
 SUMA-ENE.
     ADD CANTIDAD TO TOT-ENE-ART-1.
     GO TO LEE-FICHAS.
 SUMA-FEB.
     ADD CANTIDAD TO TOT-FEB-ART-1.
     GO TO LEE-FICHAS.
 SUMA-MAR.
     ADD CANTIDAD TO TOT-MAR-ART-1.
     GO TO LEE-FICHAS.
```

```
    SUMA-ABR.
       ADD CANTIDAD TO TOT-ABR-ART-1.
       GO TO LEE-FICHAS.
   SUMA-MAY.
       ADD CANTIDAD TO TOT-MAY-ART-1.
       GO TO LEE-FICHAS.
   SUMA-JUN.
       ADD CANTIDAD TO TOT-JUN-ART-1.
       GO TO LEE-FICHAS.
   SUMA-JUL.
       ADD CANTIDAD TO TOT-JUL-ART-1.
       GO TO LEE-FICHAS.
   SUMA-AGO.
       ADD CANTIDAD TO TOT-AGO-ART-1.
       GO TO LEE-FICHAS.
   SUMA-SEP.
       ADD CANTIDAD TO TOT-SEP-ART-1.
       GO TO LEE-FICHAS.
   SUMA-OCT.
       ADD CANTIDAD TO TOT-OCT-ART-1.
       GO TO LEE-FICHAS.
   SUMA-NOV.
       ADD CANTIDAD TO TOT-NOV-ART-1.
       GO TO LEE-FICHAS.
   SUMA-DIC.
       ADD CANTIDAD TO TOT-DIC-ART-1.
   LEE-FICHAS.
       READ FICHA INTO FICHA-WS
                  AT END
                  DISPLAY 'TOTAL VENTAS ART.1 ENERO = '
                          TOT-ENE-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 FEBRERO = '
                          TOT-FEB-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 MARZO = '
                          TOT-MAR-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 ABRIL = '
                          TOT-ABR-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 MAYO = '
                          TOT-MAY-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 JUNIO = '
                          TOT-JUN-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 JULIO = '
                          TOT-JUL-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 AGOSTO = '
                          TOT-AGO-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 SEPTIEMBRE = '
                          TOT-SEP-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 OCTUBRE = '
                          TOT-OCT-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 NOVIEMBRE = '
                          TOT-NOV-ART-1
                          UPON PRINTER
                  DISPLAY 'TOTAL VENTAS ART.1 DICIEMBRE = '
                          TOT-DIC-ART-1
                          UPON PRINTER
                  GO TO FIN-PROCESO.
       GO TO CONTROL.
   FIN-PROCESO.
       CLOSE FICHAS.
       STOP RUN.
```

ESQUEMA DE LA MEMORIA CON 12 TOTALIZADORES SEPARADOS



REPRESENTACION DE LA TABLA TAB-TOT-MENSUALES



Elemento TOTAL-MENSUAL (1)

DESCRIPCION DE UNA TABLA EN DATA DIVISION

```
01 nombre-tabla
   05 nombre-elemento        PIC... USAGE... OCCURS N TIMES
```

que contiene 12 elementos de nombre genérico TOTAL-MENSUAL, el contador del mes de ABRIL se direccionará como TOTAL-MENSUAL (4), es decir, como el campo TOTAL-MENSUAL que ocupa la cuarta posición en el ámbito de la tabla.

Una tabla es un dato compuesto (o estructurado), en el que los datos componentes, no necesariamente elementales, tienen todos el mismo nombre y pueden direccionarse sólo en base a su posición. La situación que se determina se ha esquematizado en la segunda figura de esta página.

Naturalmente, el Compilador debe saber que un cierto dato compuesto constituye una tabla. Esta información la proporciona el programador en DATA DIVISION en el acto de la descripción del campo, según el formato representado en la última tabla de arriba.

Con esta descripción, el Compilador puede reservar un área de memoria de nombre «nombre-tabla» constituida por una secuencia de N elementos «nombre-elemento». La ocupación de memoria es igual a la que se tendría si se definiesen separadamente N campos, pero utilizando la tabla TAB-TOT-MENSUALES en lugar de los doce contadores TOT-ENE-ART-1, TOT-FEB-ART-1, etc. El programa del ejemplo anterior resultará seguramente menos incómodo de detallar y más fácil de leer gracias a la drástica reducción del número de instrucciones presentes (ver el listado de las págs. 1093 y 1094).

El ejemplo representado en el listado contiene

LECTURA Y PROCESO DE UN FILE (3)

```
WORKING-STORAGE SECTION.
01 FICHA-WS.
    05 SERIE-ARTICULO           PIC 9.
    05 CODIGO-ARTICULO          PIC 9.
    05 FECHA-VENTA.
        10 AÑO                  PIC 9(2).
        10 MES                  PIC 9(2).
        10 DIA                  PIC 9(2).
    05 CANTIDAD                 PIC 9(3).
    05 COSTO-UNITARIO           PIC 9(3).
    05 FILLER                   PIC X(66).
*
*
* - - - - - - - - - - - - - - TABLAS - - - - - - - - - - - - - -
*
*** TABLA TOTALIZADORES CANTIDADES MENSUALES ART. 1 ****
*
*
01 TAB-TOT-MENSUALES.
    05 TOTAL-MENSUAL            PIC 9(6) COMP OCCURS 12.
**** TABLA DE LOS NOMBRES DE LOS MESES ****
*
*
01 MESES.
    05 FILLER                   PIC X(10) VALUE 'ENERO     '.
    05 FILLER                   PIC X(10) VALUE 'FEBRERO   '.
    05 FILLER                   PIC X(10) VALUE 'MARZO     '.
    05 FILLER                   PIC X(10) VALUE 'ABRIL     '.
    05 FILLER                   PIC X(10) VALUE 'MAYO      '.
    05 FILLER                   PIC X(10) VALUE 'JUNIO     '.
    05 FILLER                   PIC X(10) VALUE 'JULIO     '.
    05 FILLER                   PIC X(10) VALUE 'AGOSTO    '.
    05 FILLER                   PIC X(10) VALUE 'SEPTIEMBRE'.
    05 FILLER                   PIC X(10) VALUE 'OCTUBRE   '.
    05 FILLER                   PIC X(10) VALUE 'NOVIEMBRE '.
    05 FILLER                   PIC X(10) VALUE 'DICIEMBRE '.
01 TABLA-MESES REDEFINES MESES.
    05 NOMBRE-MES               PIC X(10) OCCURS 12.
*
01 INDICE-MES                   PIC 9(2) COMP.
*
PROCEDURE DIVISION.
MAIN SECTION.
INICIO.
    MOVE LOW-VALUES TO TAB-TOT-MENSUALES.
    OPEN INPUT FICHAS.
PRIMERA-LECTURA.
    READ FICHAS INTO FICHA-WS
            AT END
            DISPLAY '** FILE FICHAS VACIO **'
                    UPON PRINTER


            GO TO FIN-PROCESO.
CONTROL.
    IF CODIGO-ARTICULO = 1
        ADD CANTIDAD TO TOTAL-MENSUAL (MES).
    READ FICHAS INTO FICHA-WS
            AT END
            PERFORM DISPLAY-TOTALES
            GO TO FIN-PROCESO.
    GO TO CONTROL.
FIN-PROCESO.
    CLOSE FICHAS.
    STOP RUN.
*
*
```

```
* ============== SECCIONES =============
*
*
*
 DISPLAY-TOTALES SECTION.
 DISP-TOT.
     MOVE ZEROES TO INDICE-MES.
 GIRO-DISPLAY.
     ADD 1 TO INDICE-MES.
     IF  INDICE-MES > 12
         GO TO GIRO-DISPLAY-EX.
     DISPLAY ' TOTAL VENTAS ART. 1 '
             NOMBRE-MES (INDICE-MES)
             ' = '
             TOTAL-MENSUAL  (INDICE-MES)
             UPON PRINTER.
     GO TO GIRO-DISPLAY.
 GIRO-DISPLAY-EX. EXIT.
*
```

algunas importantes nociones que es interesante subrayar.

Para evitar escribir, al final del proceso, 12 instrucciones DISPLAY con los mensajes

TOTAL VENTAS ART. 1 MES =
total-del-mes

se ha empleado otra tabla en la que cada elemento contiene el nombre de uno de los doce meses del año. De esta manera, el mensaje a presentar en la impresora se convierte en paramétrico, y puede repetirse 12 veces indicando cada vez el nombre del mes y el total de las ventas correspondientes.

La descripción de una tabla no puede contener la cláusula VALUE. Cuando es necesario memorizar en los elementos de una tabla valores constantes y predefinidos se recurre a la técnica utilizada en el ejemplo, o sea:

1 / se define un campo compuesto de tantos subcampos como elementos de la tabla debe haber (en este caso 12).
También puede observarse que para los subcampos no es necesario utilizar nombres definidos por el programador; basta con usar el nombre reservado FILLER.

2 / se describe un campo-tabla con cláusulas REDEFINES del campo anterior, y se define el nombre de los elementos y su número en la tabla utilizando la cláusula OCCURS.

De esta manera es posible direccionar con la técnica de las tablas un elemento genérico el cual, gracias a la cláusula REDEFINES, comparte la parte de memoria del campo redefinido y, en consecuencia, asume su valor implantado.

En la primera figura de la página siguiente se ha esquematizado la situación en la memoria correspondiente al campo MESES y a la tabla TABLA-MESES que lo redefine.

La sección DISPLAY-TOTALES, llamada al final de la lectura del file FICHAS mediante el verbo PERFORM, utiliza el campo INDICE-MES para posicionarse simultáneamente en los elementos de la tabla TABLA-MESES y TAB-TOT-MENSUALES.

El valor de INDICE-MES se incrementa en 1 hasta 12, y para cada valor asumido por el índice se toma el nombre del mes correspondiente (por ejemplo, 1 = ENERO) y el contenido de la cantidad totalizada en aquel mes, para componer el mensaje deseado.

La dinámica de la creación de los doce mensajes se indica en la segunda figura de la página siguiente.

Una particular atención debe dedicarse a los ciclos de lectura o escritura de los datos en la tabla. El Compilador no puede percatarse si un

índice suma un valor superior o inferior al otorgado para la exploración de la tabla. Por ejemplo, si en el caso de la lectura de la tabla TABLA-MESES el índice se implantase erróneamente al valor 13, y la situación en la memoria fuese la del esquema de la figura de arriba de la página siguiente, el mensaje compuesto sería

TOTAL VENTAS ART. 1 ?<A47Y/& = S78)(5

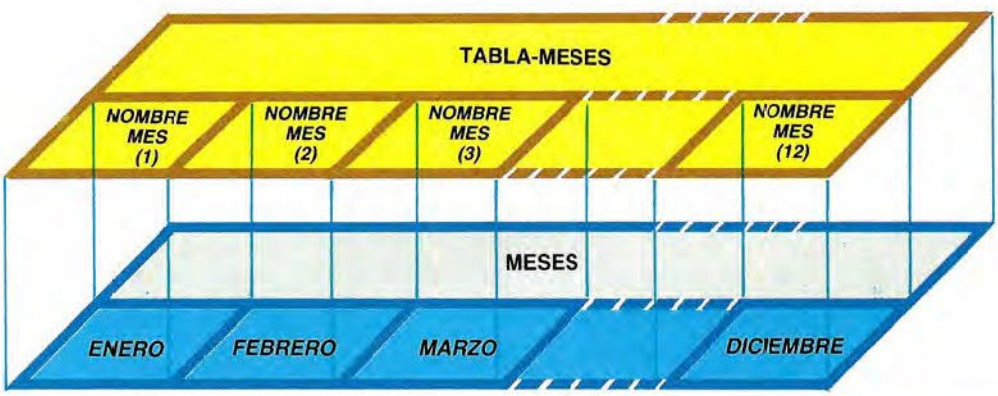
En este caso, el error se ve fácilmente, pero en otros, la gestión errónea de un índice puede conducir a una total inexactitud de los resultados o a la terminación anómala del programa.

Se volverá más ampliamente sobre el tema cuando se hable de la carga de una tabla. Por el momento, el lector debe observar la sección DISPLAY-TOTALES del ejemplo. Su diagrama de flujo, visible en la figura de abajo de la página siguiente, es extremadamente simple y, si se adopta como técnica general para la lectura de todos los datos de una tabla, permite evitar errores como los descritos antes.

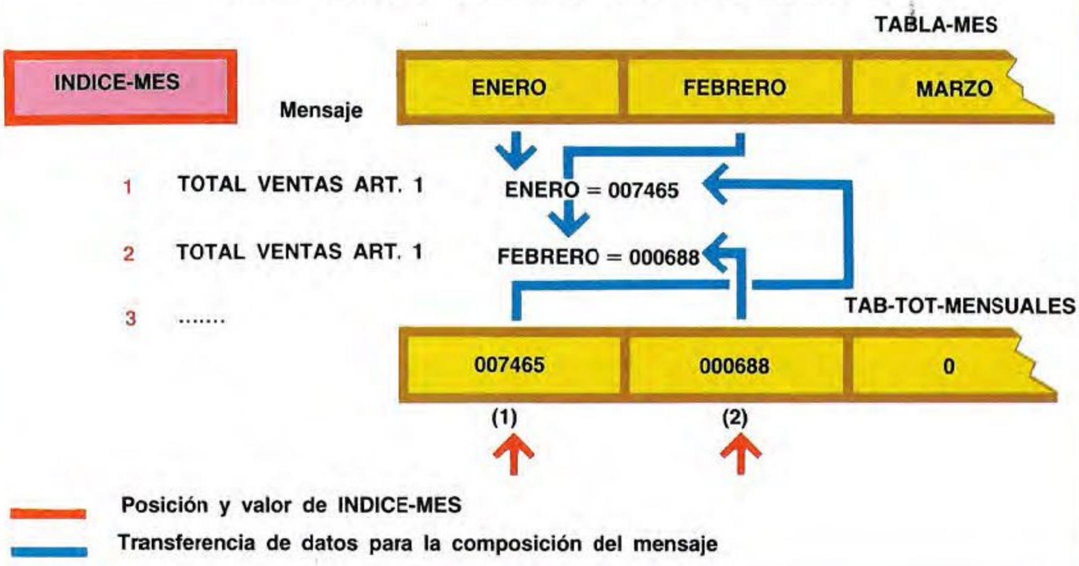
Tablas de dos dimensiones

Volviendo a considerar el ejemplo presentado al principio, puede observarse que un programa que puede procesar el total de las ventas men-

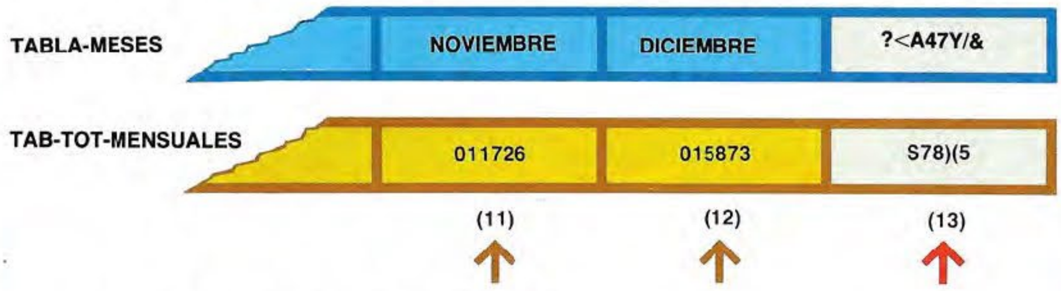
ASIGNACION DE VALORES CONSTANTES A UNA TABLA



COMPOSICION DEL CAMPO DE IMPRESION



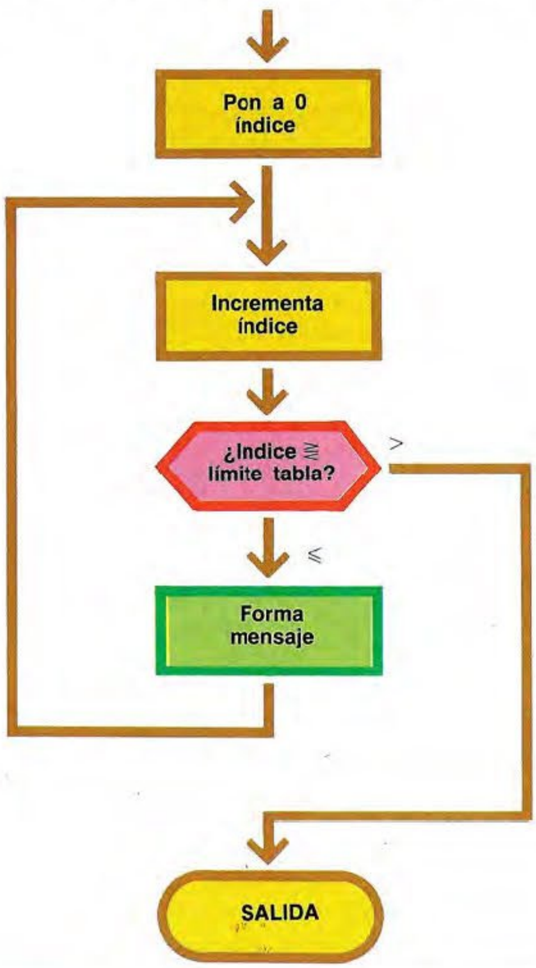
PREVENCION DE UNA TABLA PARA VALORES NO PERMITIDOS DEL INDICE



Campos adyacentes a las tablas, pero no pertenecientes a estas tablas

Posición del puntero con INDICE-MES = 13

LOGICA DE LECTURA DE UNA TABLA



suales de un solo artículo tiene una escasa utilidad. Supongamos que a todos los artículos con código hasta 9 se les quiere transferir la gestión descrita para el artículo de código 1.

Ampliando los conceptos indicados puede pensarse en gestionar la parte de memoria reservada a los totalizadores de las cantidades mensuales como se indica en la figura de abajo.

Toda la «hoja de memoria» puede considerarse como una tabla en la que cada elemento es a su vez una tabla. Por ejemplo, para leer o modificar el totalizador del mes de enero del artículo de código 2, deberá posicionarse primero en la línea número 2 con un oportuno INDICE-ARTICULO, y después mediante INDICE-MES en la primera posición (ENERO = 1) de la tabla de los totalizadores mensuales.

La situación es completamente análoga a la típica de una «batalla naval», seguramente más familiar. En este juego se comunica al adversario el propio tiro indicando los códigos de la línea y de la columna en cuyo cruce se espera encontrar un buque enemigo.

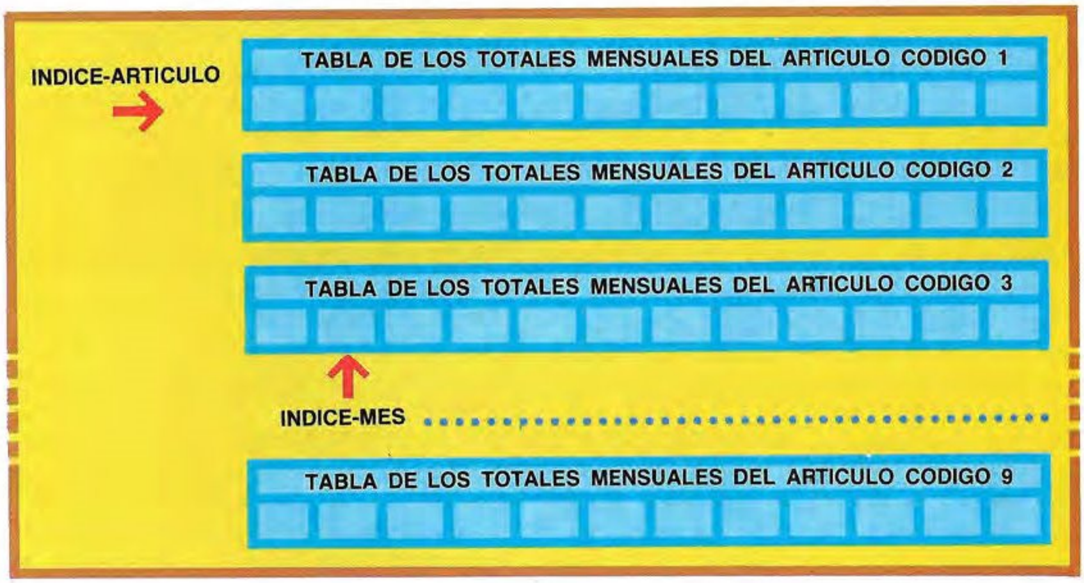
Antes de indicar el diagrama de flujo de la nueva versión del programa es oportuno detenerse en el modo de describir la «tabla de tablas» esquematizada anteriormente. Como se prevé tratar sólo 9 artículos, a cada uno de ellos deberá corresponder un elemento de la tabla. Cada elemento debe definirse después como una tabla de 12 contadores en los que totalizar las cantidades vendidas mensualmente del artículo al que se refiere la línea.

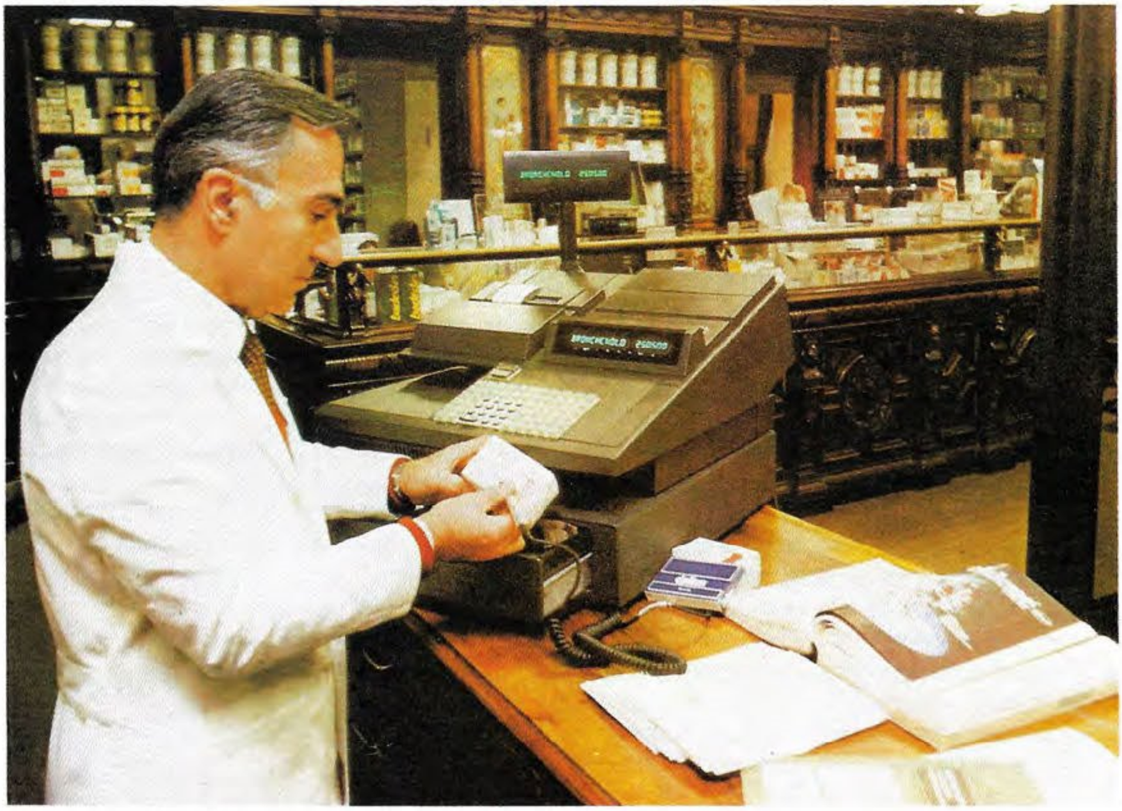
En definitiva se tendrá una ocupación de memoria de 9 líneas × 12 columnas × 4 bytes = 432 bytes. No obstante, recuérdese que un campo numérico de una longitud declarada de 5 a 9 caracteres en USAGE COMP ocupa en realidad 4 bytes. En el caso examinado, cada contador está definido como PIC 9(6) COMP y, por tanto, ocupa 4 bytes.

En la tabla de totalizadores de los artículos TAB-TOT-ART tendrá la siguiente descripción:

```
01 TAB-TOT-ART.
   05 TOTALES-ARTICULO        OCCURS-9.
      10 TAB-TOT-MENSUALES.
         15 TOTAL-MENSUAL     PIC 9(6)
                              COMP
                              OCCURS 12.
```

ESQUEMA DE LA TABLA DE DOS DIMENSIONES TAB-TOT-ART





Finsiel

Empleo del lápiz óptico conectado a una caja registradora.

El detalle de la nueva versión del programa se ha indicado en el listado de las págs. 1099 y 1100.

Aunque los totalizadores a tratar se han convertido en 9 × 12 = 108, con respecto a los 12 del ejemplo anterior, puede verse que el número de instrucciones en la MAIN SECTION ha quedado prácticamente sin alteración.

Debe hacerse otra observación a propósito de las posibilidades que ofrecen las tablas de ser inicializadas en cada uno de sus elementos utilizando una sola instrucción, cuando están constituidas por campos homogéneos.

La línea MOVE LOW-VALUES TO TAB-TOT-ART, refiriéndose al nivel más alto de la tabla (01), inicializa a LOW-VALUES todos los elementos de nivel inferior.

La función del procedimiento DISPLAY-TOTALES, llamado al final de la lectura de file FICHAS, es leer todo el contenido de la tabla de totalizadores, presentando los totales de las ventas de cada mes frente a cada artículo.

La dinámica de la lectura puede deducirse del diagrama de flujo representado en la figura de la pág. 1101.

En dicha figura se presentan las instrucciones para la lectura «horizontal», o sea de los contadores mensuales correspondientes al mismo artículo, y para las del posicionado «vertical», es decir, de línea.

Por ejemplo, obsérvese que el posicionado en el totalizador de mayo correspondiente al artículo de código 9 se efectúa llamando el elemento

TOTAL-MENSUAL (9, 5)

El modo correcto para referenciar un elemento de una tabla de tablas (o mejor dicho, de una tabla de dos dimensiones) es el siguiente

nombre-elemento (índice-línea, ƀ índice-columna)

Obsérvese el blank (espacio) que debe haber entre la coma y el índice de columna.

La impresión producida por el programa considerado se ha representado en el listado de la pág. 1102, en el que por brevedad sólo son visibles las representaciones correspondientes a los artículos de códigos 1, 2, 3 y 4.

EMPLEO DE UNA TABLA DE DOS DIMENSIONES

```
 WORKING-STORAGE SECTION.
 01 FICHA-WS.
     05 SERIE-ARTICULO                    PIC 9.
     05 CODIGO-ARTICULO                   PIC 9.
     05 FECHA-VENTA.
         10 AÑO                           PIC 9(2) COMP.
         10 MES                           PIC 9(2) COMP.
         10 DIA                           PIC 9(2) COMP.
     05 CANTIDAD                          PIC 9(3).
     05 COSTO-UNITARIO                    PIC 9(3).
     05 FILLER                            PIC X(66).
*
*
* - - - - - - - - - - - - - - TABLAS - - - - - - - - - - - - - -
*
**** TABLA TOTALIZADORES CANTIDAD MENSUAL POR ARTICULO ****
*
*
 01 TAB-TOT-ART.
     05 TOTALES-ARTICULO                  OCCURS 9.
         10 TAB-TOT-MENSUALES.
             15 TOTAL-MENSUAL             PIC 9(6) COMP OCCURS 12.
*
*
**** TABLA DE LOS NOMBRES DE LOS MESES ****
*
*
 01 MESES.
     05 FILLER                            PIC X(10) VALUE 'ENERO      '.
     05 FILLER                            PIC X(10) VALUE 'FEBRERO    '.
     05 FILLER                            PIC X(10) VALUE 'MARZO      '.
     05 FILLER                            PIC X(10) VALUE 'ABRIL      '.
     05 FILLER                            PIC X(10) VALUE 'MAYO       '.
     05 FILLER                            PIC X(10) VALUE 'JUNIO      '.
     05 FILLER                            PIC X(10) VALUE 'JULIO      '.
     05 FILLER                            PIC X(10) VALUE 'AGOSTO     '.
     05 FILLER                            PIC X(10) VALUE 'SEPTIEMBRE '.
     05 FILLER                            PIC X(10) VALUE 'OCTUBRE    '.
     05 FILLER                            PIC X(10) VALUE 'NOVIEMBRE  '.
     05 FILLER                            PIC X(10) VALUE 'DICIEMBRE  '.
 01 TABLA-MESES REDEFINES MESES.
     05 NOMBRE-MES                        PIC X(10) OCCURS 12.
*
*
 01 INDICE-ARTICULO                       PIC 9 COMP.
 01 INDICE-MES                            PIC 9(2) COMP
*
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 INICIO.
     MOVE LOW-VALUES TO TAB-TOT-ART.
     OPEN INPUT FICHAS.
 PRIMERA-LECTURA.
     READ FICHAS INTO FICHA-WS
                 AT END
                 DISPLAY '** FILE-FICHAS VACIO ** '
                          UPON PRINTER
                 GO TO FIN-PROCESO.
 SUMA.
     ADD CANTIDAD TO TOTAL-MENSUAL (CODIGO-ARTICULO, MES).
     READ FICHAS INTO FICHA-WS
                AT END
                PERFORM DISPLAY-TOTALES
                GO TO FIN-PROCESO.
     GO TO SUMA.
 FIN-PROCESO.
     CLOSE FICHAS.
     STOP RUN.
*
*
```

```
* ============== SECCIONES ==============
*
*
 DISPLAY-TOTALES SECTION.
 DISP-TOT.
      MOVE ZEROES TO INDICE-ARTICULO.
 DISPLAY-ARTICULO.
      ADD 1 TO INDICE-ARTICULO.
      IF  INDICE-ARTICULO > 9
          GO TO DISPLAY-ARTICULO-EX.
      DISPLAY SPACES UPON PRINTER.
      DISPLAY 'ARTICULO COD. '
              INDICE-ARTICULO
              '    '
              '* * * TOTALES VENTAS MENSUALES * * *'
              UPON PRINTER.
      DISPLAY SPACES UPON PRINTER.
      MOVE ZEROES TO INDICE-MES.
 DISPLAY-VENTAS.
      ADD 1 TO INDICE-MES.
      IF  INDICE-MES > 12
          GO TO DISPLAY-ARTICULO.
      DISPLAY '
              NOMBRE-MES (INDICE MES)
              ' = '
              TOTAL-MENSUAL (INDICE-ARTICULO, INDICE-MES)
              UPON PRINTER.
      GO TO DISPLAY-VENTAS.
 DISPLAY-ARTICULO-EX. EXIT.
```

Tablas de tres dimensiones

De todo lo visto hasta ahora sobre las tablas puede observarse que en la TABLA-MESES basta con un solo índice para referenciar completamente el elemento que contiene la descripción del mes. Y viceversa, para direccionar completamente el totalizador de la cantidad vendida del artículo de código 2 en el mes de mayo, deben utilizarse dos índices, con los que se posiciona primeramente en la línea correspondiente al artículo (2) y en el ámbito de ésta sobre la columna del mes de mayo (5).

En el primer caso (TABLA-MESES) se trata de tablas de una dimensión, mientras que en el segundo se ha definido de dos dimensiones.

El Cobol permite la definición de tablas con un máximo de tres dimensiones.

En el mismo ejemplo, supóngase que se quieren totalizar las cantidades vendidas mensualmente de cada artículo (códigos de 1 a 9), que puede pertenecer a 9 series diferentes.

Supóngase para más claridad que una ficha indica los valores

SERIE = 2
ARTICULO = 5
MES = 12
CANTIDAD = 007215

y otra los siguientes

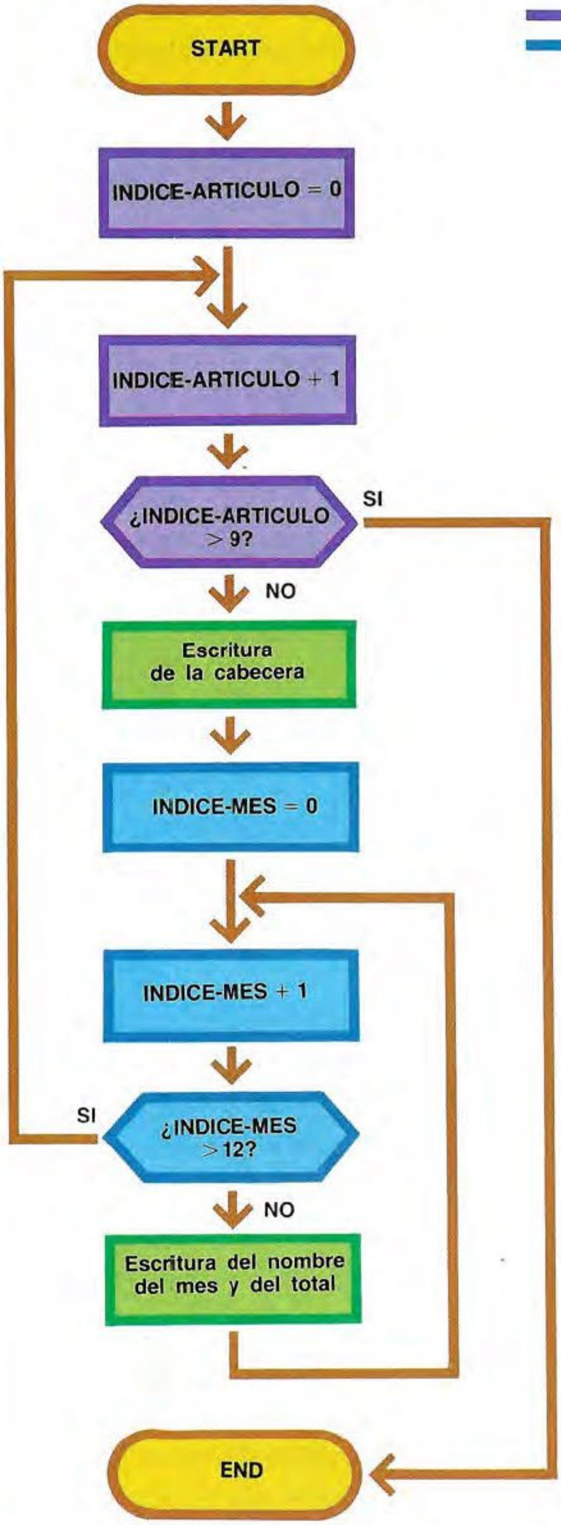
SERIE = 4
ARTICULO = 5
MES = 12
CANTIDAD = 000036

Aunque los valores de ARTICULO y MES son iguales, las cantidades deben totalizarse en dos contadores diferentes. Esta vez, el esquema de la memoria reservada a los totalizadores mensuales es el indicado en la figura de la pág. 1103. La memoria puede imaginarse como una secuencia de «hojas» direccionadas por el número de SERIE. Cada una de estas hojas se trata y direcciona después como ya se ha descrito en el ejemplo anterior, en el que cada línea corresponde a un artículo y cada campo de la línea es el totalizador de las ventas mensuales. La descripción de la tabla en WORKING-STORAGE-SECTION es la siguiente:

```
01 TAB-TOT-VENTAS.
   05 TOTALES-SERIE            OCCURS 9.
      10 TOTALES-ARTICULO      OCCURS 9.
         15 TOTAL MENSUAL      PIC 9(6)
                               COMP
                               OCCURS 12.
```

DIAGRAMA DE FLUJO DEL PROCEDIMIENTO DISPLAY-TOTALES

Instrucciones para la lectura «vertical»
Instrucciones para la lectura «horizontal»



Puesta a 0 del índice de línea INDICE-ARTICULO

Incremento del índice de línea

Control de la posición máxima «vertical» alcanzable

Se presenta la cabecera ARTICULO COD N *** TOTALES VENTAS MENSUALES ** donde N es el código genérico tratado en aquel momento

Puesta a 0 del índice de columna INDICE-MES para la lectura del contador mensual

Incremento del índice de columna

Control de la posición máxima «horizontal» alcanzable

Se presentan el nombre del mes y el total de las ventas del artículo tratado

EJEMPLO DE IMPRESION DE LOS DATOS DE UNA TABLA DE DOS DIMENSIONES

```
ARTICULO COD. 1    * * * TOTALES VENTAS MENSUALES * * *

                   ENERO      = 008974
                   FEBRERO    = 004567
                   MARZO      = 006787
                   ABRIL      = 006001
                   MAYO       = 003564
                   JUNIO      = 001234
                   JULIO      = 001234
                   AGOSTO     = 006574
                   SEPTIEMBRE = 012564
                   OCTUBRE    = 010084
                   NOVIEMBRE  = 011284
                   DICIEMBRE  = 010064


ARTICULO COD. 2    * * * TOTALES VENTAS MENSUALES * * *

                   ENERO      = 008974
                   FEBRERO    = 004567
                   MARZO      = 006787
                   ABRIL      = 006002
                   MAYO       = 003564
                   JUNIO      = 002234
                   JULIO      = 002234
                   AGOSTO     = 006574
                   SEPTIEMBRE = 022564
                   OCTUBRE    = 020084
                   NOVIEMBRE  = 021284
                   DICIEMBRE  = 020064

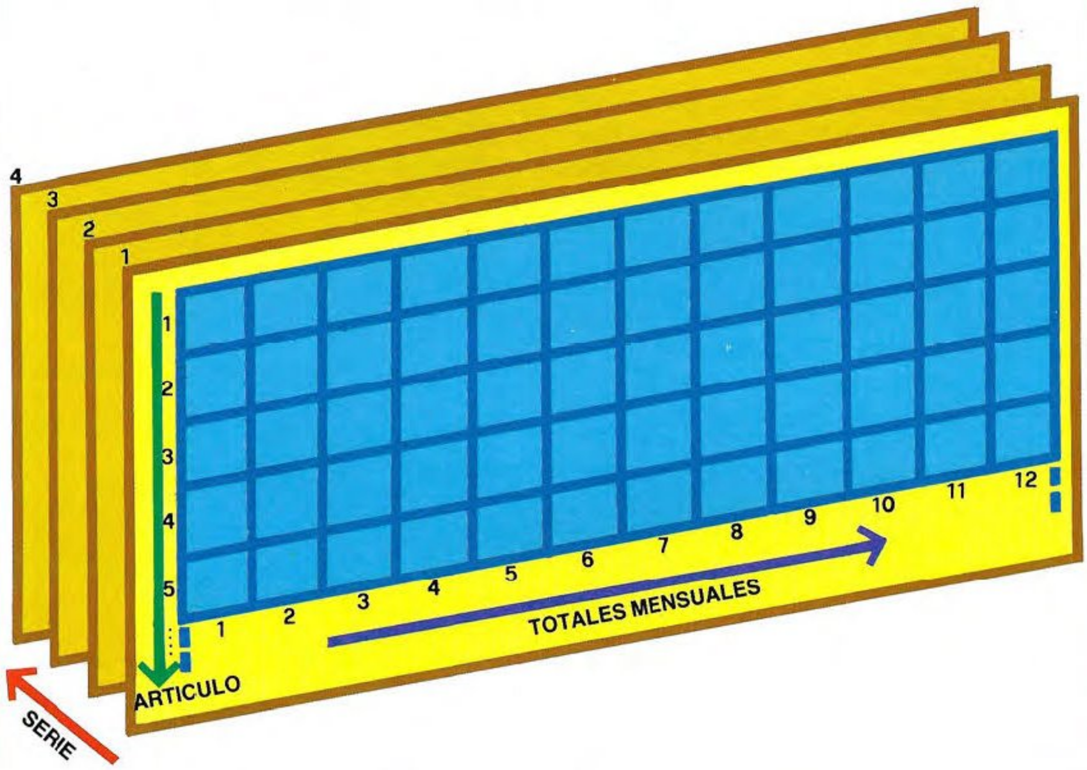

ARTICULO COD. 3    * * * TOTALES VENTAS MENSUALES * * *

                   ENERO      = 008974
                   FEBRERO    = 004567
                   MARZO      = 006787
                   ABRIL      = 006001
                   MAYO       = 003564
                   JUNIO      = 001334
                   JULIO      = 001334
                   AGOSTO     = 006574
                   SEPTIEMBRE = 013564
                   OCTUBRE    = 010084
                   NOVIEMBRE  = 011384
                   DICIEMBRE  = 010064


ARTICULO COD. 4    * * * TOTALES VENTAS MENSUALES * * *

                   ENERO      = 008974
                   FEBRERO    = 004567
                   MARZO      = 006787
                   ABRIL      = 006004
                   MAYO       = 003564
                   JUNIO      = 004234
                   JULIO      = 004234
                   AGOSTO     = 006574
                   SEPTIEMBRE = 012564
                   OCTUBRE    = 010084
                   NOVIEMBRE  = 011284
                   DICIEMBRE  = 010064
```

ESQUEMA DE LA TABLA DE TRES DIMENSIONES TAB-TOT-VENTAS



Carga de una tabla

Observando la impresión visible en el listado de la página anterior, puede verse que la dicción 'ARTICULO COD. 1' comporta, por parte del usuario, la ulterior consulta de un índice en el cual se ha indicado la descripción del artículo de código 1.

Esta circunstancia, ya criticable en el caso de pocos artículos, es inaceptable cuando la cantidad de datos es notable.

Por tanto, es preciso que las consultas del índice con el fin de apurar la correspondencia entre los códigos y las descripciones del artículo las realice el mismo programa, lo que se pide después de producir la siguiente cabecera:

ARTICULO: descripción-artículo

La correspondencia puede establecerse con una tabla en la que cada elemento contenga

1 / el número de serie (1 carácter)
2 / el código artículo (1 carácter)
3 / la descripción del artículo (30 caracteres)

Como es razonable pensar que deberán introducirse nuevos artículos y variarse algunos parámetros de los que ya están catalogados, es interesante que la carga de la tabla de descripciones se realice desde el exterior en cada ejecución del programa, por ejemplo con el empleo de un file de fichas.

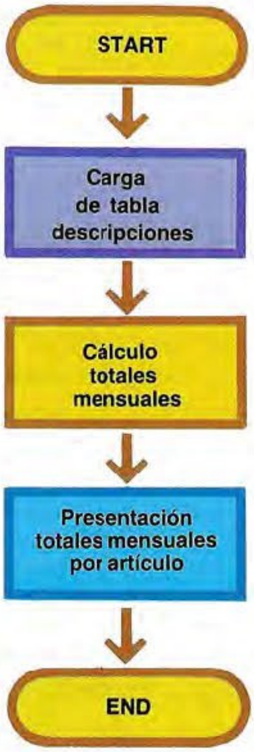
Si se adoptase la técnica ya vista para las tablas de los nombres de los meses, cada alteración del contenido de la tabla comportaría la corrección de la correspondiente descripción en WORKING-STORAGE-SECTION y la sucesiva recompilación del programa.

Sin embargo, debe indicarse que, mientras que en aquel caso podía tenerse un direccionamiento directo del contador que interesa gracias a los códigos de serie, artículo o mes, la carga de

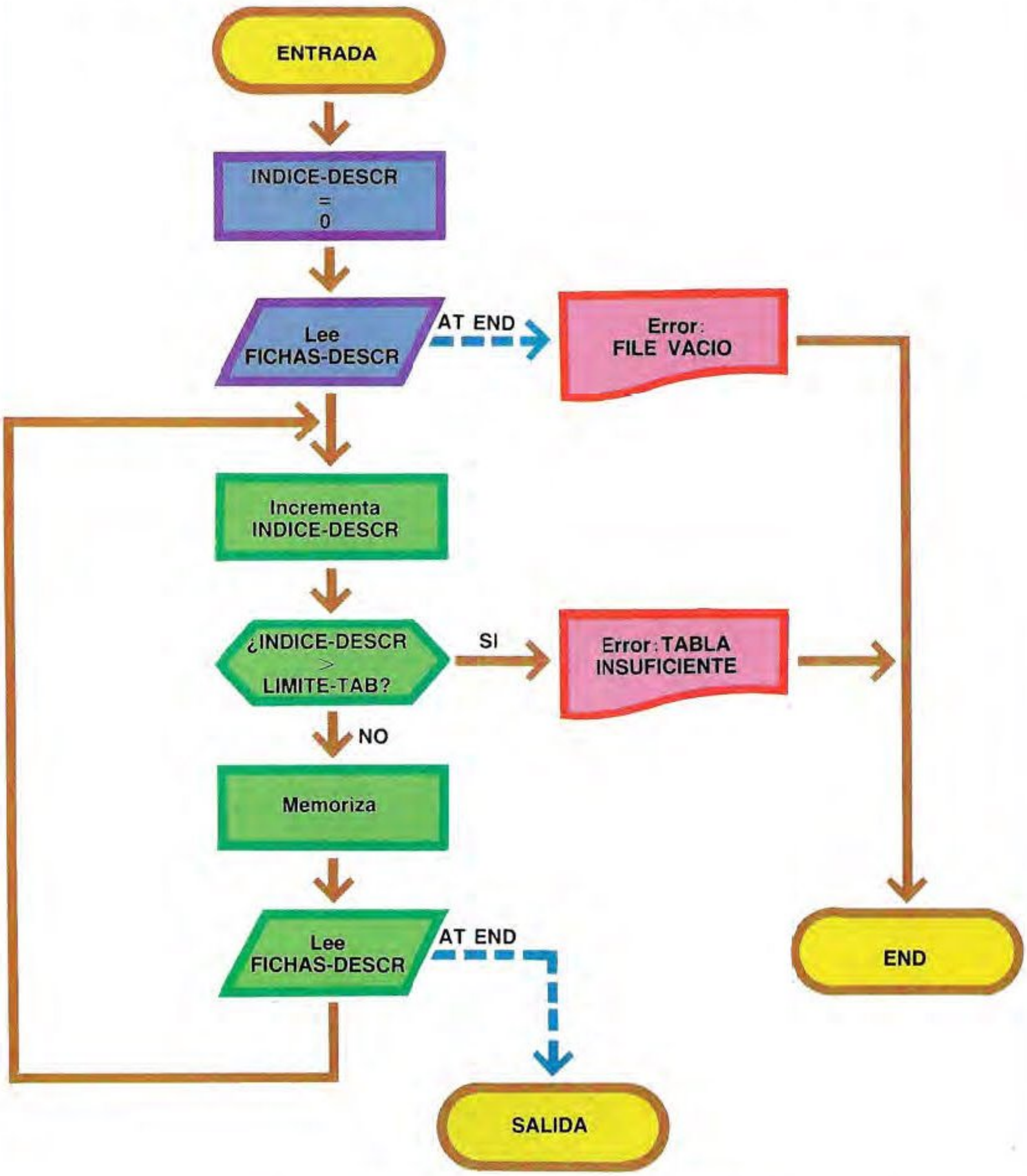
una tabla impone una gestión del índice de tipo diferente. El índice o los índices de la tabla a cargar deben gestionarse y controlarse en cada paso para obtener un llenado sin discontinuidades del espacio disponible.
Un importante control al que debe someterse el índice es el necesario para evitar el «desfondado» de la tabla.
Todo esto resultará más claro con el examen del ejemplo que sigue. En la figura de abajo se presenta el diagrama de flujo de primer nivel del programa integrado con la carga de la tabla de las descripciones de los artículos. Cada bloque puede detallarse a su vez como se ilustra en las figuras de las págs. 1105, 1106 y 1107.
Las fichas que contienen las descripciones de los artículos se suponen dispuestas en el interior del file en orden creciente de serie y artículo.
Además se supone que hay las descripciones de los nueve artículos de todas las nueve series.
Suponiendo que los artículos tratados sean prendas de vestir, el file de las fichas-descripción puede organizarse como sigue:

11 CAMISAS HOMBRE
12 CAMISAS MUJER
13 CAMISAS NIÑO
14 ....
15 ....
..
..
21 PANTALONES HOMBRE
22 PANTALONES MUJER
23 PANTALONES NIÑO
24 ....
25 ....
..
..
..
91 ZAPATOS HOMBRE
92 ZAPATOS MUJER
..
..
99 ZAPATITOS BEBE

PROGRAMA QUE UTILIZA LA CARGA Y LA GESTION DE UNA TABLA FLOW-CHART



FASE DE CARGA DE LA TABLA DE LAS DESCRIPCIONES



En el diagrama de flujo representado en la página siguiente puede verse que la técnica de lectura de la tabla todavía es la de antes, o sea:

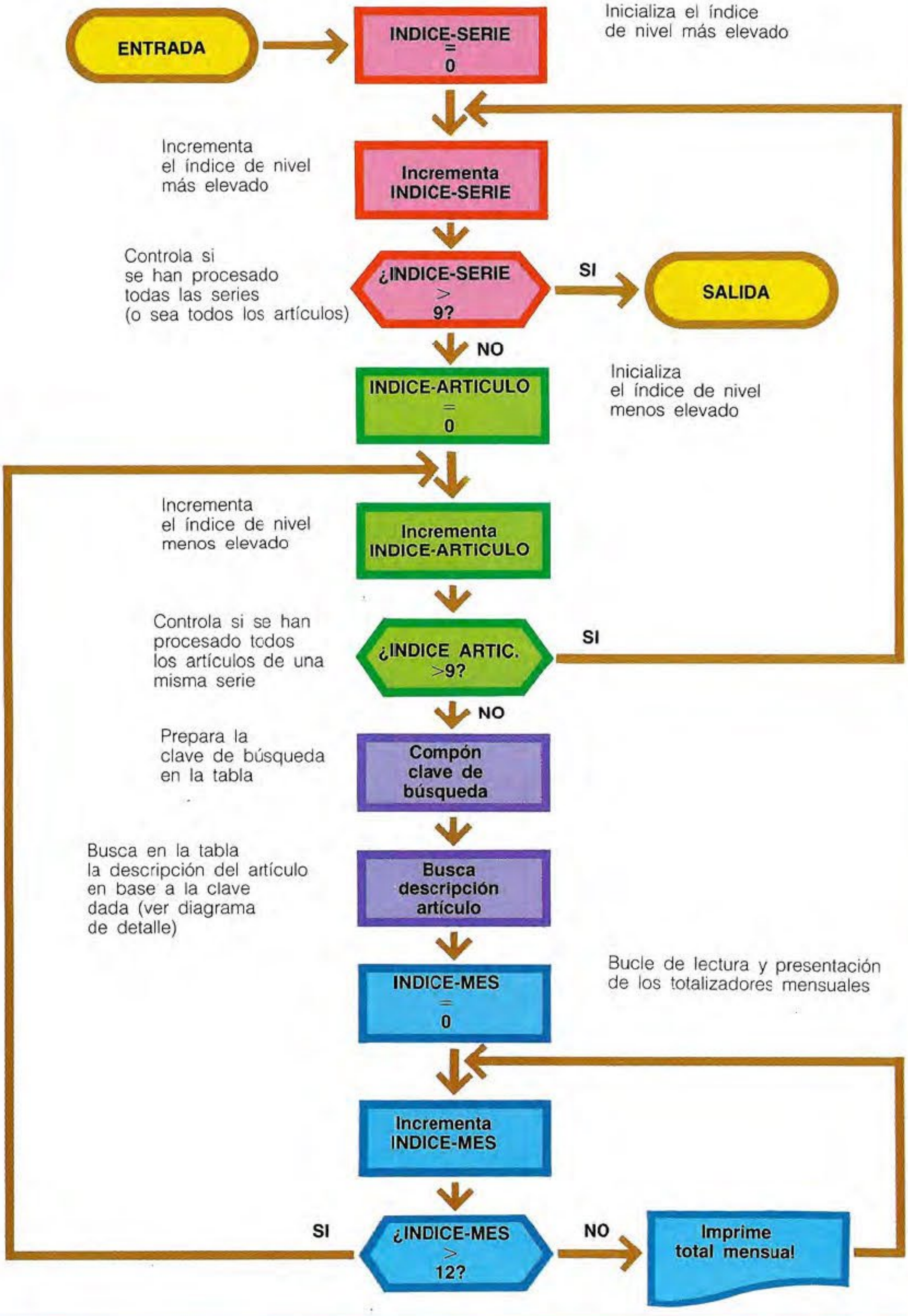
1 / se incrementa y controla el índice de nivel más elevado (INDICE-SERIE)

2 / el valor de este índice no se altera hasta que no se han tratado todos los artículos (control por INDICE-ARTICULO mayor que 9)

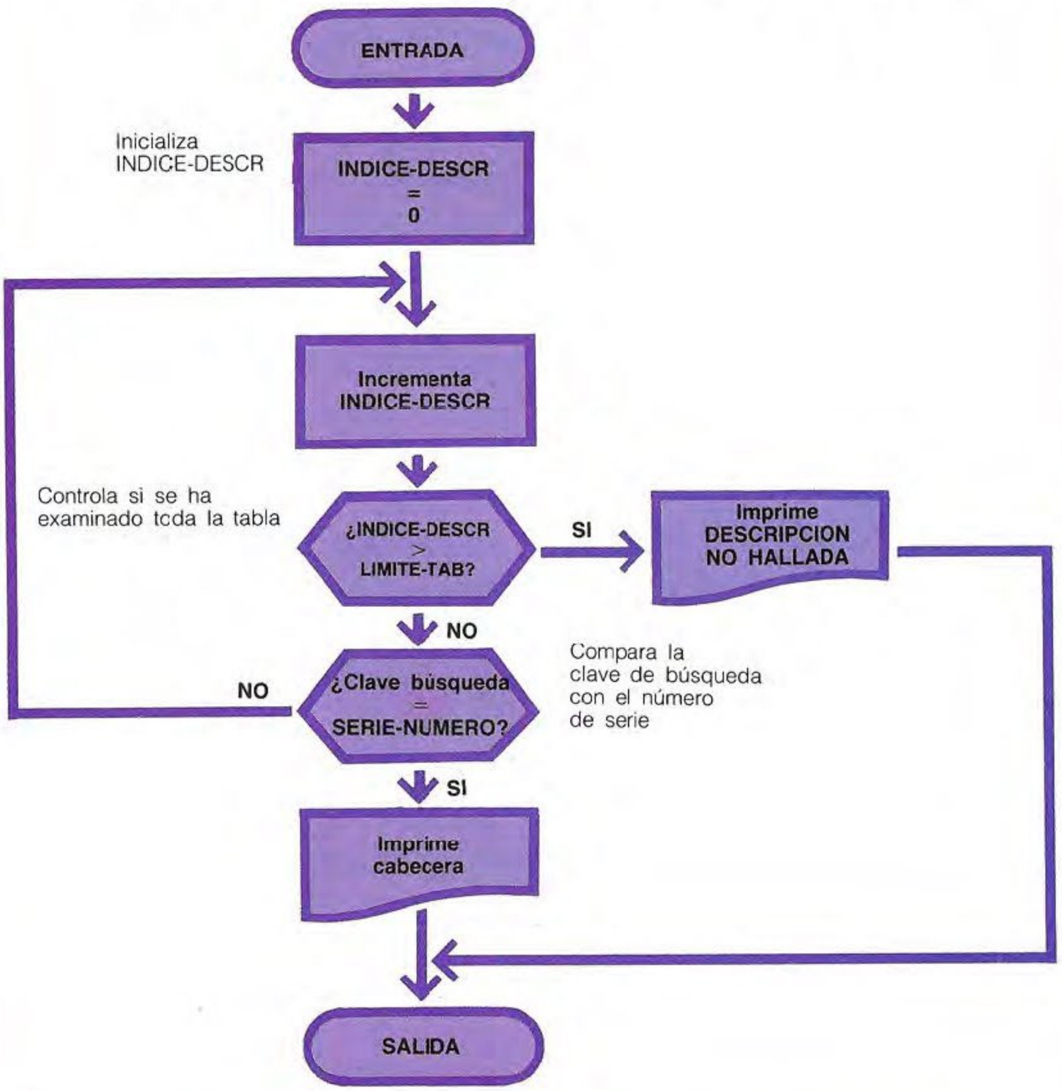
3 / para cada valor del índice de artículo se efectúa un bucle para leer y presentar el contenido de todos los totalizadores mensuales.

Debe darse un particular relieve a la parte dedicada a la búsqueda en la tabla, efectuada para extraer la descripción de cada artículo (figura de la pág. 1107). Como se intenta presentar la descripción mercantil de todos los artículos, pa-

FASE DE LECTURA DE LAS FICHAS-DESCRIPCION



FASE DE BUSQUEDA DE LA DESCRIPCION DE UN ARTICULO



ra cada uno de éstos se ha compuesto un campo auxiliar (clave de búsqueda) con los valores del número de serie y del código del artículo.
Después, el campo clave se compara iterativamente con los dos primeros caracteres de cada elemento de la tabla de las descripciones.
El bucle se interrumpe cuando los dos campos comparados resultan iguales; en este caso se toma la parte DESCRIPCION del elemento para componer la línea de cabecera.
Es interesante prever el caso en que la igualdad descrita no se verifique nunca. Para ello se ha previsto la sustitución de la descripción del artículo por su códigos identificadores (serie y número) acompañados de la señalización DESCRIPCION NO HALLADA.
El programa descrito se ha representado en las págs. 1108, 1109 y 1110. Se han omitido las descripciones de algunas partes, para las cuales se remite al lector a los ejemplos anteriores.

Los índices de las tablas

Para direccionar un campo en una tabla se han utilizado hasta ahora campos definidos en WORKING-STORAGE SECTION. Piénsese por ejemplo en NOMBRE-MES y en INDICE-MES

PROGRAMA QUE UTILIZA LA CARGA Y LA GESTION DE UNA TABLA

```
 *
 *
 *
 DATA DIVISION.
 FILE SECTION.
 FD  FICHAS-DESCR
     LABEL RECORD OMITTED.
 01  FICHA-DESCR                PIC X(80).
 FD  FICHAS
     LABEL RECORD OMITTED.
 01  FICHA                      PIC X(80).
 *
 *
 WORKING-STORAGE SECTION.
 01  FICHA-DESCR-WS.
     05  SERIE-SK               PIC 9.
     05  CODIGO-SK              PIC 9.
     05  DESCRIPCION-SK         PIC X(30).
     05  FILLER                 PIC X(48).
 *
 *
 01  FICHA-WS.
     05  SERIE-ARTICULO         PIC 9.
     05  CODIGO-ARTICULO        PIC 9.
     05  FECHA-VENTA.
         10  AÑO                PIC 9(2)  COMP.
         10  MES                PIC 9(2)  COMP.
         10  DIA                PIC 9(2)  COMP.
     05  CANTIDAD               PIC 9(3).
     05  COSTO-UNITARIO         PIC 9(3).
     05  FILLER                 PIC X(66).
 *
 *
 * -------------- TABLAS ----------------
 *
 *
 01  TAB-TOTALES-VENTAS.
     05  TOTALES-SERIE          OCCURS 9.
         10  TOTALES-ARTICULO   OCCURS 9.
             15 TOTAL-MENSUAL    PIC 9(6) COMP OCCURS 12.
 *
 *
 **** TABLA DE LOS NOMBRES DE LOS MESES ****
 *
 *
 01  MESES.
     05  FILLER                 PIC X(10)  VALUE 'ENERO     '.
     05  FILLER                 PIC X(10)  VALUE 'FEBRERO   '.
     05  FILLER                 PIC X(10)  VALUE 'MARZO     '.
     05  FILLER                 PIC X(10)  VALUE 'ABRIL     '.
     05  FILLER                 PIC X(10)  VALUE 'MAYO      '.
     05  FILLER                 PIC X(10)  VALUE 'JUNIO     '.
     05  FILLER                 PIC X(10)  VALUE 'JULIO     '.
     05  FILLER                 PIC X(10)  VALUE 'AGOSTO    '.
     05  FILLER                 PIC X(10)  VALUE 'SEPTIEMBRE'.
     05  FILLER                 PIC X(10)  VALUE 'OCTUBRE   '.
     05  FILLER                 PIC X(10)  VALUE 'NOVIEMBRE '.
     05  FILLER                 PIC X(10)  VALUE 'DICIEMBRE '.
 01  TABLA-MESES  REDEFINES  MESES.
     05  NOMBRE-MES             PIC X(9)  OCCURS 12.
 *
 *
 *        TABLA DE LAS DESCRIPCIONES DE LOS ARTICULOS
 *
 *
 01  TABLA-DESCR.
     05  DESCR-ART              OCCURS 100.
```

```
          10   CLAVE-ELEMENTO.
               15  SERIE-ELEM        PIC 9.
               15  CODIGO-ELEM       PIC 9.
          10   DESCRIPCION           PIC X(30).
*
*                 = = = INDICES = = =
*
*
 01  INDICE-ARTICULO                 PIC 9     COMP VALUE 0.
 01  INDICE-SERIE                    PIC 9     COMP VALUE 0.
 01  INDICE-MES                      PIC 9(2)  COMP VALUE 0.
 01  INDICE-DESCR                    PIC 9(2)  COMP VALUE 0.
 01  LIMITE-TAB-DESCR                PIC 9(3)  COMP VALUE 100.
*
*
*                 = = = AUXILIARES = = =
*
 01  CLAVE.
     05  SERIE-CLAVE                 PIC 9.
     05  CODIGO-CLAVE                PIC 9.
*
*
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 ABRE-FILE-DESCR.
     OPEN INPUT FICHAS-DESCR.
 PRIMERA-LECTURA-DESCR.
     READ FICHA-DESCR INTO FICHA-DESCR-WS
                    AT END
                    DISPLAY '** FILE DESCRIPCIONES VACIO **'
                          UPON PRINTER
                    CLOSE FICHAS-DESCR
                    GO TO FIN PROCESO.
 GIRO-DESCRIPCIONES
     ADD 1 TO INDICE-DESCR.
     IF  INDICE-DESCR > LIMITE-TAB-DESCR
         CLOSE FICHAS-DESCR
         DISPLAY '** TABLA DESCRIPCIONES INSUFICIENTE **'
               UPON PRINTER
         DISPLAY 'MAXIMO NUMERO DESCRIPCIONES PREVISTO = 100'
               UPON PRINTER
         GO TO FIN-PROCESO.
     MOVE FICHA-DESCR-WS TO DESCR-ART (INDICE-DESCR).
     READ FICHAS-DESCR INTO FICHA-DESCR-WS
                  AT END
                  CLOSE FICHAS-DESCR
                  DISPLAY '** DESCRIPCIONES CARGADAS = '
                        INDICE-DESCR
                        UPON PRINTER
                  GO TO CARGA-DATOS.
     GO TO GIRO-DESCRIPCIONES.
 CARGA-DATOS.
     OPEN INPUT FICHAS.
     MOVE LOW-VALUES TO TAB-TOTALES-VENTAS.
 PRIMERA-LECTURA-DATOS.
     READ FICHAS INTO FICHA-WS
                 AT END
                 CLOSE FICHAS
                 DISPLAY '** FILE FICHAS DATOS VACIO **'
                       UPON PRINTER
                 GO TO FIN-PROCESO.
 SUMA.
     MOVE SERIE-ARTICULO    TO INDICE-SERIE.
     MOVE CODIGO-ARTICULO   TO INDICE-ARTICULO.
     MOVE MES               TO INDICE-MES.
     ADD CANTIDAD TO TOTAL-MENSUAL (INDICE-SERIE, INDICE-ARTICULO,
        INDICE-MES).
 OTROS DATOS.
     READ FICHAS INTO FICHA-WS
                 AT END
```

```
                    CLOSE FICHAS
                    PERFORM DISPLAY-TOTALES
                    GO TO FIN-PROCESO.
    GO TO SUMA.
*
*
*
FIN-PROCESO.
    STOP RUN.
*
*
*
DISPLAY-TOTALES SECTION.
DISP-TOT.
    MOVE ZEROES TO INDICE-SERIE.
DISPLAY-SERIE.
    ADD 1 TO INDICE-SERIE.
    IF  INDICE-SERIE > 9
        GO TO DISP-TOT-EX.
    MOVE ZEROES TO INDICE-ARTICULO.
DISPLAY-ARTICULO.
    ADD 1 TO INDICE-ARTICULO.
    IF  INDICE-ARTICULO > 9
        GO TO DISPLAY-SERIE.
    MOVE ZEROES TO INDICE-DESCR.
    MOVE INDICE-SERIE    TO  SERIE-CLAVE.
    MOVE INDICE-ARTICULO TO  CODIGO-CLAVE.
BUSCA-DESCRIPCION.
    ADD 1 TO INDICE-DESCR.
    IF  INDICE-DESCR > LIMITE-TAB-DESCR
        DISPLAY 'ARTICULO SERIE   : '
                INDICE-SERIE
                'CODIGO           : '
                INDICE-ARTICULO
                ' ** DESCRIPCION NO HALLADA **'
                ' * * * TOTALES VENTAS * * *'
                UPON PRINTER
        GO TO ESCRIBE-TOTALES.
    MOVE INDICE-SERIE    TO SERIE-CLAVE.
    MOVE INDICE-ARTICULO TO CODIGO-CLAVE.
    IF  CLAVE NOT = CLAVE-ELEMENTO (INDICE-DESCR)
        GO TO BUSCA-DESCRIPCION.
    DISPLAY 'ARTICULO : '
            DESCRIPCION (INDICE-DESCR)
            ' * * * TOTALES VENTAS * * * '
            UPON PRINTER.
ESCRIBE-TOTALES.
    MOVE ZEROES TO INDICE-MES.
DISPLAY-VENTAS.
    ADD 1 TO INDICE-MES.
    IF  INDICE-MES > 12
        GO TO DISPLAY-ARTICULO.
    DISPLAY '
          NOMBRE-MES (INDICE-MES)
            ' = '
            TOTAL-MENSUAL (INDICE-SERIE, INDICE-ARTICULO,
    INDICE-MES)
            UPON PRINTER.
    GO TO DISPLAY-VENTAS.
DISP-TOT-EX. EXIT.
*
*
*
```

De este modo, el Compilador trata los campos indicados en base a sus características y no en base a la función de índice que el programador le reserva. Es decir, el programador podría utilizar el campo INDICE-MES para cualquier operación permitida por el Cobol sin tener ningún error de compilación, ya que la función de índice asociada al campo no se ha declarado al Compilador.
Los campos como INDICE-MES, INDICE-ARTICULO, etc., descritos en WORKING-STORAGE SECTION y utilizados por el programador como índices, se llaman **subindexados.**
El Cobol prevé la posibilidad de asociar a una tabla uno o más índices reconocidos como tales por el Compilador y gestionados por éste de manera específica: de hecho, **un índice no requiere definiciones en WORKING-STORAGE.** Su declaración al Compilador se hace durante la descripción de la tabla, a la que el índice queda unívocamente ligado. Considérese la TABLA-MESES del ejemplo anterior. Si se desea asociar a esta tabla un índice gestionado como tal por el Compilador, debe adoptarse la notación indicada en el listado de abajo.
El campo IND-MES, asociado unívocamente a la tabla mediante la cláusula INDEXED BY, **sólo** puede usarse para direccionar los campos de TABLA-MESES. Cada uno de estos campos, sin embargo, puede direccionarse, además de con IND-MES, con un subindexado cualquiera. En síntesis, cuando la TABLA-MESES se describa como arriba, las tres instrucciones siguientes son válidas

```
MOVE NOMBRE-MES (3) TO ........
MOVE NOMBRE-MES (IND-MES) TO ........
MOVE NOMBRE-MES (INDICE-MES) TO .......
```

En la primera, el direccionado se obtiene con una constante (3), en la segunda con el índice de la tabla (IND-MES) y en la tercera con un subindexado (INDICE-MES).
Obsérvese finalmente que a una misma tabla se pueden asociar más índices, aunque manteniendo para cada uno las limitaciones descritas. Por ejemplo, podría escribirse

```
.
.
01 TABLA-MESES        REDEFINES MESES.
   05 NOMBRE-MES      PIC X(10) OCCURS 12
                      INDEXED BY
                      IND-MES
                      NUMERO-MES
                      PUNTERO
                      ......
```

El verbo SET para la gestión de un índice. El índice asociado a una tabla mediante la cláusula INDEXED BY no necesita descripción en WORKING-STORAGE, puesto que el Compilador reserva para ello un registro interno propio de formato particular que puede gestionar automáticamente cuando se usan las instrucciones que se comentarán más adelante.
Teniendo en cuenta las particularidades de este registro, el programador no puede intervenir en su contenido con las instrucciones Cobol usadas por los subindexados (MOVE, ADD ...), sino que debe utilizar la instrucción SET. El verbo SET, según el formato usado, permite

- situar el índice a un determinado valor
- incrementar a voluntad en una cantidad el valor actual del índice

DESCRIPCION DE UNA TABLA CON INDICE GESTIONADO POR EL COMPILADOR

```
*01     MESES.
        05  FILLER                      PIC X(10)   VALUE 'ENERO'.
        05  FILLER                      PIC X(10)   VALUE 'FEBRERO'.
        05  FILLER                      PIC X(10)   VALUE 'MARZO'.
        05  FILLER                      PIC X(10)   VALUE 'ABRIL'.
        05  FILLER                      PIC X(10)   VALUE 'MAYO'.
        05  FILLER                      PIC X(10)   VALUE 'JUNIO'.
        05  FILLER                      PIC X(10)   VALUE 'JULIO'.
        05  FILLER                      PIC X(10)   VALUE 'AGOSTO'.
        05  FILLER                      PIC X(10)   VALUE 'SEPTIEMBRE'.
        05  FILLER                      PIC X(10)   VALUE 'OCTUBRE.
        05  FILLER                      PIC X(10)   VALUE 'NOVIEMBRE.
        05  FILLER                      PIC X(10)   VALUE 'DICIEMBRE.
01      TABLA-MESES    REDEFINES    MESES.
        05  NOMBRE-MES                  PIC X(10)   OCCURS 12
                                        INDEXED BY IND-MES.
```

- decrementar en una determinada cantidad el valor actual del índice.

Las tres funciones indicadas se liberan respectivamente de los formatos descritos en la tabla de abajo. Debe observarse que los valores permitidos por un índice están ligados a las dimensiones de la tabla. En el caso de la tabla TABLA-MESES, el índice IND-MES sólo puede asumir los valores de 1 a 12, tantos como cuantos son los elementos declarados por tabla; los valores no comprendidos en el intervalo indicado hacen imprevisible el contenido del índice.
El verbo SET y los índices asociados pueden aparecer inútilmente redundantes para la gestión de una tabla, pero después se verán las razones de su introducción.

El verbo SEARCH para la búsqueda en las tablas

Se vuelve de nuevo al ejemplo desarrollado hasta ahora para el tratamiento de las tablas. Para poder identificar la descripción de un determinado artículo se ha compuesto una clave de búsqueda con los números de serie y de artículo. La clave se compara sucesivamente con los dos primeros caracteres de todos los 100 elementos de la tabla de las descripciones: si la clave de búsqueda es igual a la clave del elemento se toma la parte descripción y, en caso contrario, se incrementa el índice para una nueva comparación.
Este proceso se interrumpe al hallar el elemento, o bien cuando el índice supera el número de elementos que puede contener dicha tabla.
En el caso del ejemplo podía haber 9 series de 9 tipos de artículos cada una, o sea un máximo de 81 descripciones. Por tanto, es evidente que los pasos de comparación 82 al 100 son inútiles y representan un despilfarro en términos de tiempo de proceso. Por otra parte, se ha impuesto cargar en la primera fase del proceso las descripciones de todos los artículos y series posibles. Si no se hubiese introducido esta imposición, las descripciones a cargar habrían podido ser un número cualquiera menor que 81 y, por tanto, los pasos de comparación inútiles habrían sido aún más numerosos.
Poder limitar la búsqueda a los elementos efectivamente cargados sin explorar toda la tabla es particularmente ventajoso en el caso de tablas muy grandes, sobre todo si están llenas solamente con un porcentaje relativamente bajo de su capacidad.
Para poder efectuar la búsqueda de forma limitada sólo a los elementos cargados, basta con memorizar en la fase de carga el número de estos elementos en un oportuno campo auxiliar. En la siguiente fase de búsqueda, la tabla se considerará agotada cuando el índice haya superado el contenido del campo auxiliar.
El párrafo GIRO-DESCRIPCIONES del ejemplo anterior, en el que se efectuaba la carga de las descripciones de los artículos del file de fichas FICHAS-DESCR, puede reescribirse así:

```
GIRO-DESCRIPCIONES.
   ADD 1 TO INDICE-DESCR.
   IF INDICE-DESCR>LIMITE-TAB-DESCR.
      CLOSE FICHAS-DESCR
      DISPLAY 'TABLA DESCRIPCIONES
               INSUFICIENTE'
               UPON PRINTER
               GO TO FIN-PROCESO.
```

FORMATOS DE LA INSTRUCCION SET

<u>SET</u> nombre-índice <u>TO</u>	{ constante / nombre-dato / nombre-índice-1 }
<u>SET</u> nombre-índice <u>UP BY</u>	{ constante / nombre-de-dato }
<u>SET</u> nombre-índice <u>DOWN BY</u>	{ constante / nombre-de-dato }

```
    MOVE FICHA-DESCR-WS
            TO DESCR-ART (INDICE-DESCR).
    READ FICHAS-DESCR INTO
            FICHA-DESCR-WS
                AT END.
    MOVE INDICE-DESCR
            TO CUENTA-ELEMENTOS
    GLOSE FICHAS-DESCR.
    GO TO CARGA-DATOS.
    GO TO GIRO-DESCRIPCIONES.
CARGA-DATOS.
    ........................
```

Al final de la lectura del file FICHAS-DESCR, INDICE-DESCR, que contiene aún el número de posición del último elemento cargado, se salva en el campo auxiliar CUENTA-ELEMENTOS, que se supone descrito como

```
 01 CUENTA-ELEMENTOS     PIC 9(3) COMP.
```

Con estas premisas, en la sección DISPLAY-TOTALES, el párrafo BUSCA-DESCRIPCION puede reescribirse como sigue:

```
BUSCA-DESCRIPCION.
    ADD 1 TO INDICE-DESCR.
    IF INDICE-DESCR>CUENTA-ELEMENTOS
      DISPLAY 'ARTICULO SERIE'
                INDICE-SERIE
                'CODIGO'
                INDICE-ARTICULO
                *** DESCRIPCION
                NO HALLADA ***
                UPON PRINTER
      GO TO ESCRIBE-TOTALES.
...............
...............
```

Puede observarse que la descripción se considera «no hallada» en la tabla cuando se han explorado todos los elementos realmente presentes, o bien cuando el índice ha superado el valor contenido en CUENTA-ELEMENTOS.
Todo esto constituye una útil premisa al examen de la instrucción SEARCH y de las tablas de dimensión variable.

La instrucción SEARCH. La instrucción SEARCH (del inglés buscar) permite, en un primer formato, efectuar una búsqueda secuencial del tipo examinado y emprender al menos dos tipos de acciones diferentes según que el resultado de la búsqueda sea positivo o negativo.

De acuerdo con el carácter discursivo del lenguaje Cobol, la instrucción SEARCH puede traducirse en una frase normal de tipo «humano». La búsqueda de las descripciones del artículo en la tabla TABLA-DESCR equivale al siguiente comando: «busca (SEARCH) el elemento de la tabla DESCR-ART, y cuando (WHEN) el campo CLAVE-ELEMENTO sea igual a la CLAVE, escribe la descripción del artículo. Si llegas al final de la tabla sin haber encontrado igualdad (AT END) indica DESCRIPCION NO HALLADA». Adoptando la instrucción SEARCH, el programador debe

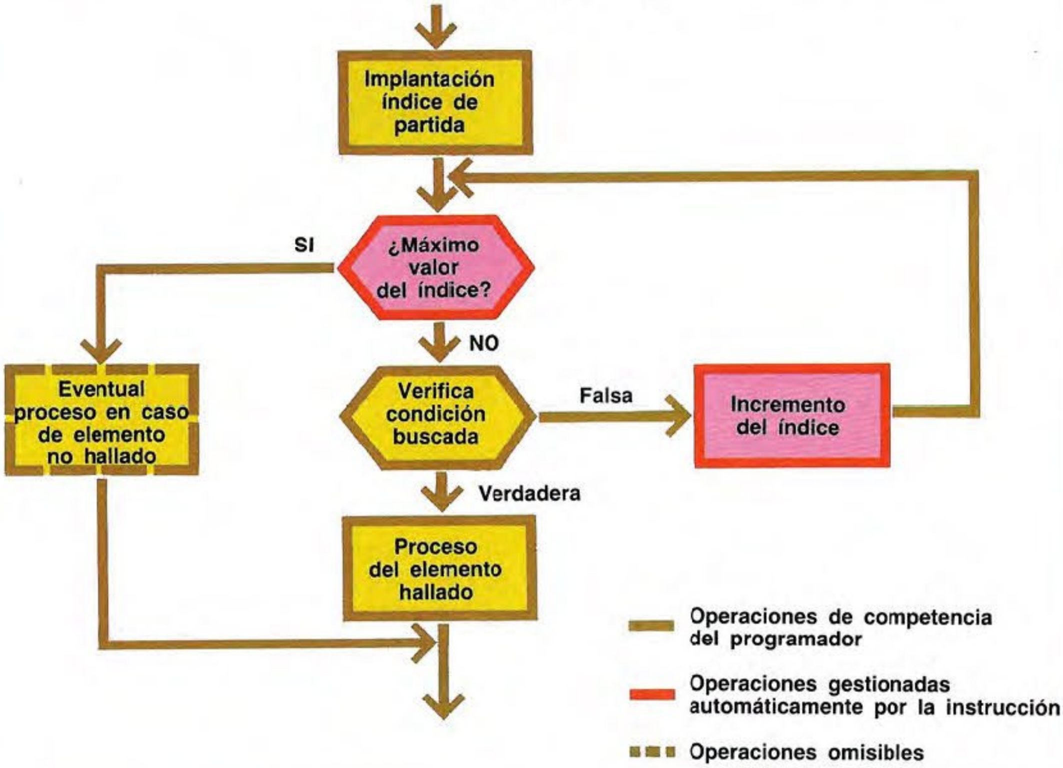
1 / implantar el valor del índice correspondiente a la posición desde la que quiere empezar la búsqueda; normalmente se explora la tabla desde el primer elemento (SET índice TO 1)
2 / especificar el tipo de acción que debe emprenderse cuando se cumple la condición de búsqueda
3 / especificar la acción a emprender en el caso en que el elemento buscado no esté en la tabla.

En el gráfico de la página siguiente se ha representado el mecanismo usado por la SEARCH secuencial.
Para evidenciar el empleo de la SEARCH secuencial pueden compararse (ver listados de las págs. 1114 y 1115) los dos formatos del párrafo BUSCA-DESCRIPCION cuando se adopta la siguiente definición para TABLA-DESCR:

```
01  TABLA-DESCR.
    05 DESCR-ART       OCCURS 100
                       INDEXED BY IND-DES.
        10 CLAVE-ELEMENTO.
            15 SERIE-ELEM          PIC 9.
            15 CODIGO-ELEM         PIC 9.
        10 DESCRIPCION             PIC X(30).
```

En la versión del párrafo en que la búsqueda la gestiona el programador con el indexado INDICE-DESCR, se utiliza el límite máximo de las dimensiones de la tabla y no el límite de rellenado (CUENTA-ELEMENTOS). Efectivamente, sólo en este caso, las dos versiones del párrafo resultan completamente equivalentes desde el punto de vista funcional.
El formato general de la instrucción SEARCH para la búsqueda del tipo secuencial se ha representado en la tabla de la pág. 1115.

ESQUEMA DE EJECUCION DE UNA INSTRUCCION SEARCH DE TIPO SECUENCIAL



BUSQUEDA EN TABLA MEDIANTE USO DE SUBINDEXADO

```
          .
          .
          .
          MOVE ZEROES TO INDICE-DESCR.
      BUSCA-DESCRIPCION.
          ADD 1 TO INDICE-DESCR.
          IF  INDICE-DESCR GREATER THAN LIMITE-TAB-DESCR
              DISPLAY 'ARTICULO SERIE '
                      INDICE-SERIE
                      'CODIGO '
                      INDICE-ARTICULO
                      ' *** DESCRIPCION NO HALLADA ***'
                      UPON PRINTER
                      GO TO ESCRIBE-TOTALES.
     *
     *
      COMPON-CLAVE.
          MOVE INDICE-SERIE      TO SERIE-CLAVE.
          MOVE INDICE-ARTICULO   TO CODIGO-CLAVE.
      COMPARA-CLAVE.
          IF CLAVE NOT = CLAVE-ELEMENTO (INDICE-DESCR)
             GO TO BUSCA-DESCRIPCION.
          DISPLAY 'ARTICULO : '
                  DESCRIPCION (INDICE-DESCR)
                  '*** TOTALES VENTAS ***'
                  UPON PRINTER.
     *
     *
      ESCRIBE-TOTALES.
          ....................
```

BUSQUEDA EN TABLA MEDIANTE SEARCH SECUENCIAL

```
 BUSCA-DESCRIPCION.
      SET IND-DES TO 1.
      MOVE INDICE-SERIE          TO SERIE-CLAVE.
      MOVE INDICE-ARTICULO       TO CODIGO-CLAVE.
*
*
      SEARCH DESCR-ART
             AT END DISPLAY 'ARTICULO SERIE
                            INDICE-SERIE
                            ' CODIGO '
                            INDICE-ARTICULO
                            ' *** DESCRIPCION NO HALLADA ***'
                            UPON PRINTER
             WHEN CLAVE-ELEMENTO (INTO-DES) = CLAVE
                    DISPLAY 'ARTICULO : '
                            DESCRIPCION (IND-DES)
                            ' *** TOTALES VENTAS ***'
                            UPON PRINTER.
*
*
 ESCRIBE-TOTALES.
      ....................
```

FORMATO GENERAL DE LA INSTRUCCION SEARCH

```
SEARCH nombre-elemento [VARYING {nombre-índice}] [AT END {frase-imperativa-1}]
                                {nombre-dato  }          {NEXT SENTENCE     }

        WHEN condición-1 {frase-imperativa-2}
                         {NEXT SENTENCE     }

      [ WHEN condición-2 {frase-imperativa-3} ]  ......................
                         {NEXT SENTENCE     }
```

Nombre-elemento es el nombre genérico del elemento de la tabla en que debe hacerse la búsqueda. Cuando no se especifica la cláusula AT END, si la condición del elemento hallado no se verifica nunca, el control pasa a la primera instrucción que sigue al punto de cierre de la SEARCH.

Uso de la SEARCH secuencial. El problema de no continuar la exploración de la tabla más allá del límite del llenado real de la misma, ya indicado en la búsqueda mediante indexados, puede resolverse de varias maneras adoptando la SEARCH.

Un primer método es el que puede verse en la figura de la página siguiente.

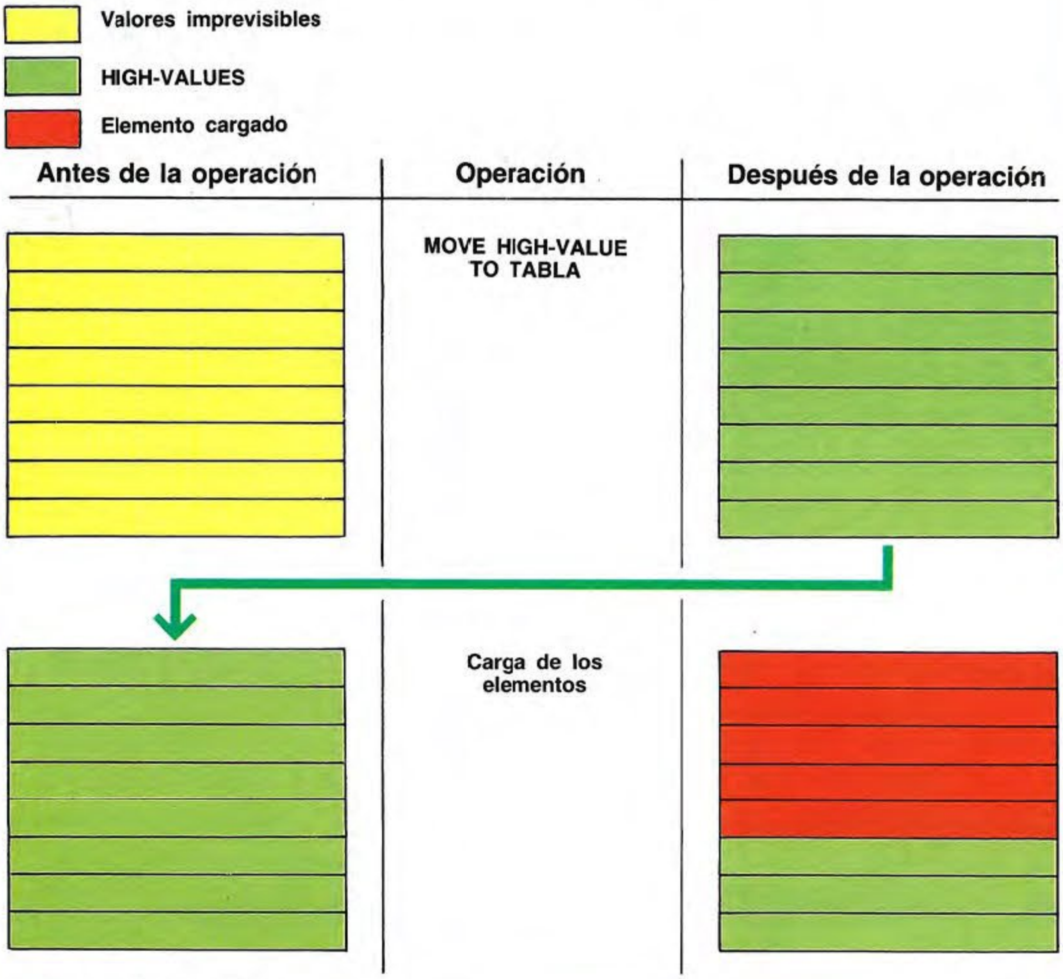
Se implantan HIGH-VALUE en todos los elementos de la tabla y después se cargan los elementos reales. La condición de fin de búsqueda se producirá si la tabla se termina o si el último elemento examinado es igual a HIGH VALUE.

Los pasos a efectuar en el ámbito del programa para limitar la búsqueda sólo a los elementos realmente presentes en la tabla pueden sintetizarse como se indica en dicha figura.

Un segundo método para minimizar los tiempos de búsqueda secuencial aprovecha las posibilidades previstas por el Cobol para declarar al Compilador el número de los elementos sobre los que efectuar la búsqueda.

Este segundo método requiere la definición de la tabla interesada como tabla de dimensiones variables, en la que el número de elementos constituyentes es específico del valor de un

ESQUEMA DE LA TABLA EN MEMORIA DURANTE LA CARGA




```
PROCEDURE DIVISION.
    .
    .
    .
        MOVE HIGH-VALUES   TO  TABLA.
        PERFORM CARGA-TABLA.
          .
          .
          .
PROCESA.
          .
          .
          .
BUSCA-ELEMENTO.
        SEARCH ELEMENTO
               AT END
               PERFORM NO-HALLADO
        WHEN ELEMENTO (INDICE) = HIGH-VALUES
               PERFORM NO-HALLADO
        WHEN CLAVE-ELEMENTO (INDICE) = CLAVE-BUSQUEDA
               PERFORM PROCESA-ELEMENTO.
```

DESCRIPCION EN WORKING-STORAGE SECTION DE UNA TABLA DE DIMENSION VARIABLE

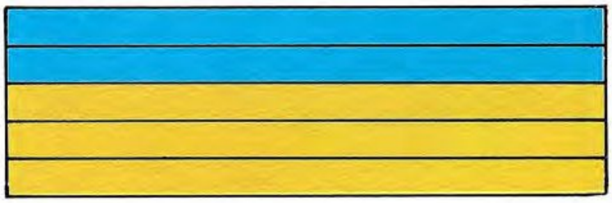
```
 01  DIMENSION                 PIC 9  COMP.
*
 01  TABLA.
     05  ELEMENTO              PIC X(10)
                               OCCURS 1 TO 5
                               DEPENDING ON DIMENSION
                               INDEXED BY IND-TAB.
```



Representación esquemática del espacio físico reservado a la tabla

MOVE 2 TO DIMENSION

Limitación a 2 elementos de la dimensión de la tabla



Esquema del espacio físico reservado a la tabla y de la parte (establecida por DIMENSION = 2) sobre la que actúan las instrucciones (por ejemplo, SEARCH)

campo numérico definido por el programador. La descripción de este tipo de tabla requiere los siguientes parámetros.

- el número mínimo de elementos que la tabla puede contener
- el número máximo de elementos que la tabla puede contener
- el nombre del campo numérico cuyo valor establece las dimensiones reales de la tabla.

La descripción de una tabla de dimensiones variables debe atenerse al siguiente formato:

```
01 TABLA.
   05 ELEMENTO   OCCURS mínimo
                 TO máximo TIMES
                 DEPENDING ON
                 nombre-de-campo.
```

«mínimo» debe ser una constante mayor que 0, sin signo y menor que «máximo»

«nombre-de-campo» es el nombre de un campo numérico que puede asumir sólo valores comprendidos entre «mínimo» y «máximo».

Es interesante aclarar que la dicción «tabla de dimensiones variables» no debe interpretarse como posibilidad de expandir dinámicamente el espacio ocupado en la memoria durante la carga, ya que el Compilador reserva una parte de memoria suficiente para contener la tabla dimensionada al máximo.

La variabilidad de las dimensiones de la tabla en función del contenido de «nombre-de-dato» es puramente lógica, en el sentido de que una instrucción cualquiera que trate la tabla en un determinado momento sólo ve la parte declarada del valor nombre-de-dato.

La situación se ha esquematizado en el ejemplo de arriba.

Volviendo ahora a considerar el programa del ejemplo, puede intuirse fácilmente que el método más rápido para buscar secuencialmente la

BUSQUEDA SECUENCIAL EN UNA TABLA

```
 WORKING-STORAGE SECTION.
  .
  .
  .
 01   CUENTA-ELEMENTOS                       PIC 9(3)   COMP VALUE 0.
 01   TABLA-DESCR.
      05   DESCR-ART                         OCCURS 1 TO 100
                                             DEPENDING ON CUENTA-ELEMENTOS
                                             INDEXED BY IND-DES.
           10   CLAVE-ELEMENTO.
                15   SERIE-ELEM              PIC 9.
                15   CODIGO-ELEM             PIC 9.
           10   DESCRIPCION                  PIC X(30).
 01   INDICE-DESCR                           PIC 9(3)   COMP VALUE 0.
 01   LIMITE-TAB-DESCR                       PIC 9(3)   COMP VALUE 100.
 01   CLAVE.
      05   SERIE-CLAVE                       PIC 9.
      05   CODIGO-CLAVE                      PIC 9.
*
*
*
 PROCEDURE DIVISION.
  .
  .
 CARGA-DESCRIPCIONES.
      ADD   1   TO   INDICE-DESCR.
      IF   INDICE-DESCR > LIMITE-TAB-DESCR
           CLOSE FICHAS-DESCR
           DISPLAY 'TABLA DESCRIPCIONES INSUFICIENTE'
                     UPON PRINTER
           GO TO FIN PROCESO.
      MOVE FICHA-DESCR-WS TO DESCR-ART (INDICE-DESCR).
      READ FICHAS-DESCR INTO FICHA-DESCR-WS
                                AT END
                                MOVE INDICE-DESCR TO CUENTA-ELEMENTOS.
                                CLOSE FICHAS-DESCR
                                GO TO CARGA-DATOS.
      GO TO CARGA DESCRIPCIONES.
  .
  .
  .
  .
*
*
*
 CARGA-DATOS.
 ......................
 ......................
 ......................
  .
  .
  .
*
*
*
 BUSCA-DESCRIPCION.
      SET   IND-DES             TO 1.
      MOVE  INDICE-SERIE        TO SERIE-CLAVE.
      MOVE  INDICE-ARTICULO     TO CODIGO-CLAVE.
      SEARCH DESCR-ART
             AT END
                    DISPLAY 'ARTICULO SERIE '
                                INDICE-SERIE
                                ' CODIGO ' ,
                                INDICE-ARTICULO
                                '*** DESCRIPCION NO HALLADA ***'
      UPON PRINTER
WHEN
      CLAVE-ELEMENTO (IND-DES) = CLAVE
```

```
                              DISPLAY 'ARTICULO : '
                                      DESCRIPCION (IND-DES)
                                      '  *** TOTAL VENTAS ***'
                                      UPON PRINTER.
      "
      "
      "
 *
 *
 *
  ESCRIBE TOTALES.
  " " " " " " " " " " " " " " " "
```

descripción de un artículo es el indicado en el listado de la página anterior y de arriba, en el que se han omitido las partes no interesadas directamente en la aplicación.

La instrucción SEARCH ALL (búsqueda binaria). De todo lo ilustrado hasta ahora sobre la búsqueda en las tablas puede observarse que el uso de la SEARCH de tipo secuencial comporta ventajas relativamente modestas con respecto a una búsqueda mediante indexados gestionada por el programador. Ahora veremos un formato diferente de la instrucción SEARCH, mucho más rápido que el anterior.

Para la aplicación de la instrucción SEARCH ALL se presupone el ordenado ascendente o descendente de los elementos de la tabla con respecto a un campo clave declarado.

Para aclarar el concepto se volverá a la tabla de las descripciones de los artículos. Las especificaciones del programa preveían que los files de fichas de las descripciones (FICHAS-DESCR) estuviesen en orden creciente de serie y artículo, como se indica a continuación:

Clave de comparación

Serie	Artículo	Descripción
└→11	←┘CAMISAS	HOMBRE
12	CAMISAS	MUJER
13	CAMISAS	NIÑO
.		
21	PANTALONES	HOMBRE
22	PANTALONES	MUJER
23	PANTALONES	NIÑO
.		
91	ZAPATOS	HOMBRE
92	ZAPATOS	MUJER
.		
99	ZAPATITOS	BEBE

Así, después de la carga, la tabla TABLA-DESCR respetará el ordenado indicado.

Al realizar una búsqueda de tipo secuencial, por ejemplo del artículo que tiene serie 2 y código artículo 3, o sea una clave compuesta 23, la clave de búsqueda CLAVE se compara primero con la clave del primer elemento (11), después con la del segundo (12) y así sucesivamente hasta el duodécimo elemento, cuya clave es 23. Con esta técnica, el número de accesos a efectuar en la tabla para verificar la existencia de un elemento es en promedio igual a N/2, donde N es el número de elementos sobre los que se hace la búsqueda.

Es claro que el número de accesos realizados en la tabla dependerá de la posición efectiva del elemento buscado, en el sentido de que la instrucción accederá, por ejemplo, una sola vez si el elemento deseado está en la primera posición, mientras que el acceso deberá repetirse N veces si el elemento buscado está en la última posición de la tabla.

La lógica de búsqueda de la SEARCH ALL es análoga a la utilizada en el juego del «alto y bajo». El jugador N.1 debe adivinar un número comprendido entre 1 y 100 escrito por el jugador N.2 en una hoja de papel.

El jugador N.1 puede hacer intentos a los que el jugador N.2 sólo puede responder con frases «demasiado grande» o «demasiado pequeño», dando así indicaciones de posición.

En la tabla de la página siguiente se ha representado el proceso del juego, suponiendo que el número a adivinar es 47.

En la figura de la pág. 1121 se ha esquematizado la situación correspondiente a los diversos intentos efectuados.

Como puede observarse, mientras que una búsqueda secuencial habría comportado 47 intentos antes de obtener el resultado, la búsqueda binaria sólo ha necesitado 5 accesos. Este tipo de búsqueda, llamada también **búsqueda**

Jugador N. 1	Comentario	Jugador N. 2
50	El jugador N. 1 trata de saber rápidamente en qué mitad de la serie de 1 a 100 está el número, efectuando el primer intento con el número central.	Alto
25	La respuesta (Alto) indica que el número se encuentra entre los primeros 50, por lo que se dejarán los números 51 a 100. Siguiendo el razonamiento que ha llevado a la primera conclusión, debe averiguarse en qué mitad de la serie de 1 a 50 está el número.	Bajo
37	Ahora, el jugador N. 1 sabe que el número está entre 25 y 50, por lo que divide de nuevo el intervalo por la mitad.	Bajo
44	La última respuesta del jugador N. 2 indica que el número está entre 37 y 50, por lo que el jugador N. 1 suma al número del último intento (37) la mitad del intervalo entre 37 y 50, o sea 7.	Bajo
47	El número debe estar entre 44 y 50. El intento, efectuado siempre con el mismo criterio, es el último, puesto que da con el número exacto.	Exacto

dicotómica (del griego corta por la mitad) es la adoptada por la SEARCH ALL. Por tanto, se intuye el motivo por el cual la tabla debe ordenarse en sentido ascendente o descendente con respecto a la clave de búsqueda.

En este punto debe precisarse que el ordenado de la tabla está completamente a cargo del programador, ya que el Compilador no emprende ninguna acción en este sentido: después de cada intento infructuoso, el Compilador debe saber qué parte de la tabla queda excluida de los siguientes intentos. Esta precisión es obligada, ya que el formato con que debe describirse una tabla que debe someterse a SEARCH ALL podría inducir a engaño.

Efectivamente, este formato prevé la declaración del tipo de ordenado aplicado **por el programador** a la tabla, y puede resumirse como se ha indicado en la tabla de abajo.

La cláusula ASCENDING/DESCENDING KEY IS clave-de-búsqueda declara al Compilador el tipo de ordenado aplicado a la tabla y la parte del elemento que debe utilizarse como clave de búsqueda. Para fijar ideas se vuelve al ejemplo de la tabla TABLA-DESCR que contiene las descripciones de los artículos.

Para aplicar en fase de búsqueda de las descripciones la cláusula SEARCH ALL, la definición de la tabla y el párrafo BUSCA-DESCRIPCION pueden detallarse como se indica en el listado de la pág. 1122.

El formato de la instrucción SEARCH ALL se describe en la tabla de la página siguiente.

Obsérvese que, contrariamente al caso de la SEARCH binaria, no se tiene necesidad de la implantación inicial del índice (SET índice TO...), ya que éste está gestionado completamente por la propia instrucción.

Además, la única comparación posible con la SEARCH ALL entre la clave de búsqueda y la del elemento en la tabla es por igualdad.

DESCRIPCION DE UNA TABLA A SOMETER A SEARCH ALL

```
01 nombre-tabla
   05 nombre-elemento    OCCURS mínimo TO máximo
                         DEPENDING ON dimensión

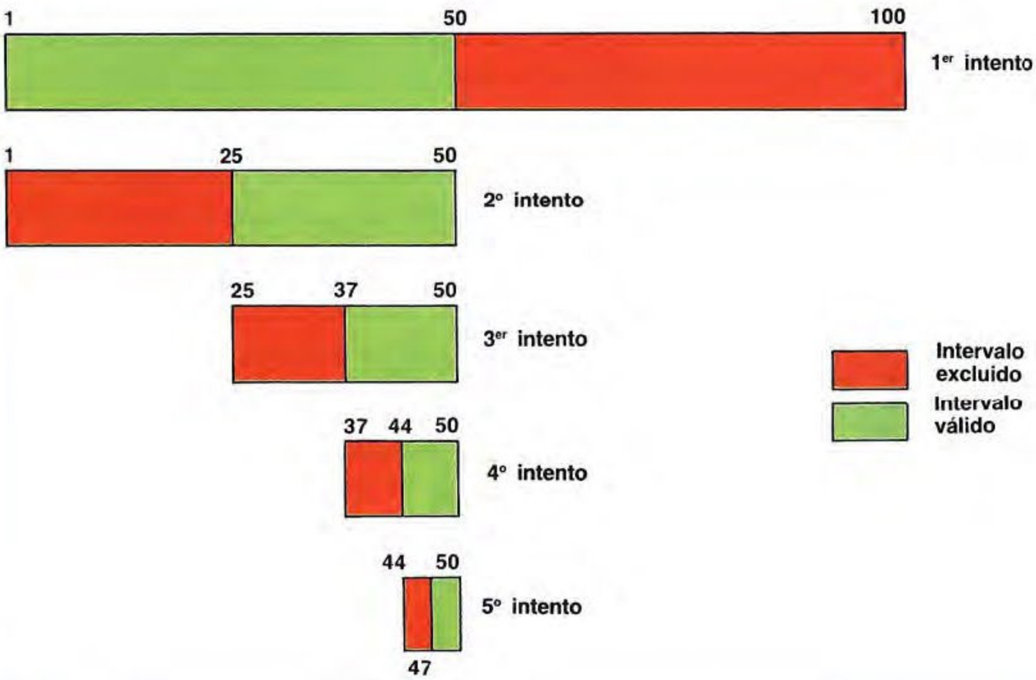
                         { ASCENDING  }
                         { DESCENDING }  KEY IS clave-de-búsqueda

                         INDEXED BY nombre-índice
```

FORMATO DE LA INSTRUCCION SEARCH ALL

```
SEARCH ALL nombre-elemento
        AT END
                    { frase-imperativa-1 }
                    { NEXT-SENTENCE      }
                                            { constante        }
        WHEN        clave-de-búsqueda =     { nombre-de-campo  }
                                            { expresión        }
                    { frase-imperativa-2 }
                    { NEXT-SENTENCE      }
```

BUSQUEDA DICOTOMICA DEL NUMERO 47 EN EL INTERVALO 1 ÷ 100



Esta limitación no existe para la SEARCH de tipo secuencial, en la que se admiten comparaciones por mayor y menor.

Ordenado de una tabla

En el uso del Cobol, el problema de la consulta de una tabla es muy frecuente. Los motivos por los que se recurre a la tabulación de una serie de datos y a su ulterior consulta pueden ser muy diversos, según las necesidades del programa. El tema desarrollado sobre la búsqueda en las tablas y sobre los instrumentos estándar del Cobol orientados a aquélla (SEARCH y SEARCH ALL) ha evidenciado las ventajas de la manipulación de datos mediante tablas.

Dejando aparte los casos más sencillos de tablas de valores fijos, el análisis correspondiente a la utilización de una tabla, especialmente si es de grandes dimensiones, debe realizarse atentamente, y más si el calculador empleado no es particularmente rápido.

La SEARCH ALL permite acelerar los procesos de búsqueda en las tablas reduciendo drásticamente el número de accesos, pero requiere que

BUSQUEDA DICOTOMICA EN UNA TABLA

```
 01   TABLA-DESCR.
      05  DESCR-ART                    OCCURS 1 TO 100
                                       DEPENDING ON CUENTA-ELEMENTOS
                                       ASCENDING KEY IS CLAVE-ELEMENTO
                                       INDEXED BY IND-DES.
          10  CLAVE-ELEMENTO.
              15  SERIE-ELEM           PIC 9.
              15  CODIGO-ELEM          PIC 9.
          10  DESCRIPCION              PIC X(30).
*
*
*
      .
      .
      .
*
*
 BUSCA-DESCRIPCION.
      MOVE INDICE-SERIE      TO  SERIE-CLAVE.
      MOVE INDICE-ARTICULO   TO  CODIGO-CLAVE.
      SEARCH ALL DESCR-ART
                 AT END DISPLAY 'ARTICULO SERIE '
                                INDICE-SERIE
                                ' CODIGO '
                                INDICE-ARTICULO
                                ' *** DESCRIPCION NO HALLADA ***'
                                UPON PRINTER
                 WHEN CLAVE-ELEMENTO (IND-DES) = CLAVE
                      DISPLAY 'ARTICULO : '
                              DESCRIPCION (IND-DES)
                              ' *** TOTALES VENTAS ***'
                              UPON PRINTER.
*
*
 ESCRIBE-TOTALES.
      ..................



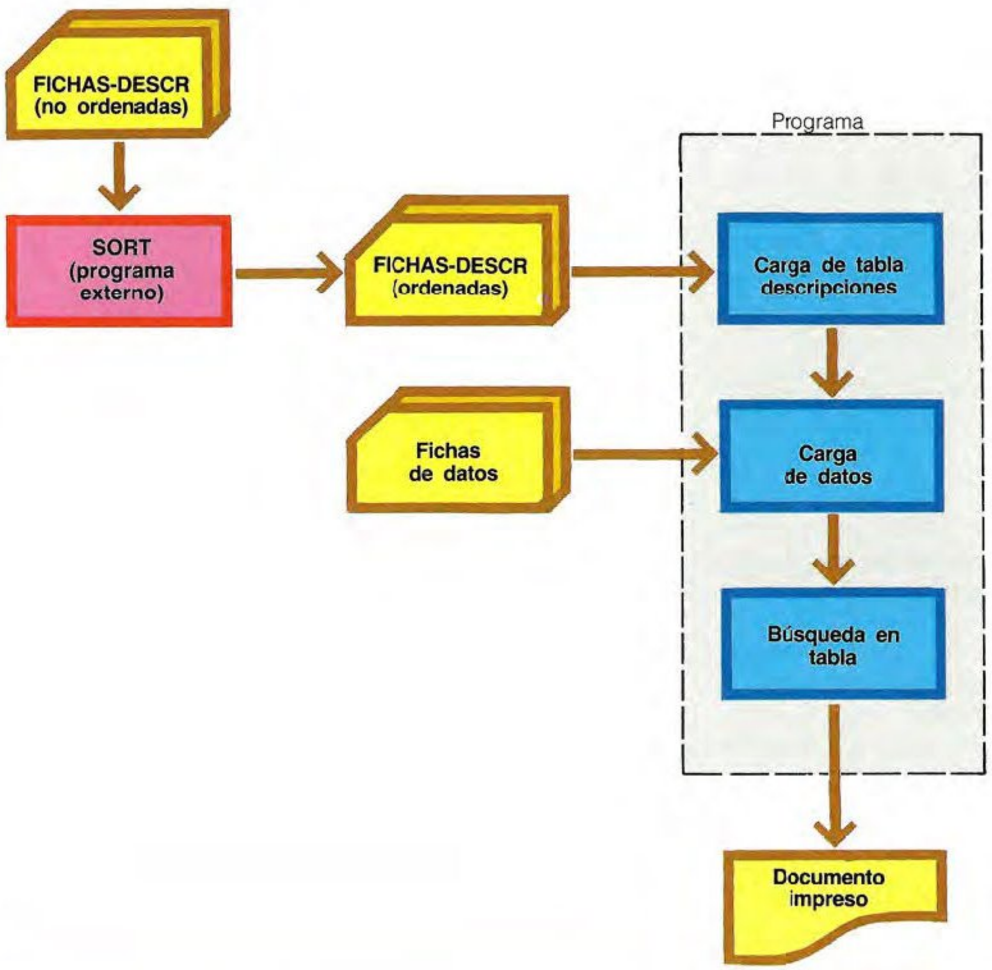

      ..................
      ..................
```

la tabla se haya ordenado previamente según una determinada clave.
Si los datos a cargar en la tabla no están seleccionados ya según un determinado orden, creciente o decreciente, se plantea el problema de si conviene adoptar una búsqueda de tipo secuencial o de tipo binario. Si se desea utilizar este último es necesario prever una fase de ordenación de los datos, que puede realizarse de tres modos diferentes.
El file de datos puede ordenarse según la clave deseada utilizando los adecuados programas (Sort) de que están provistos la mayoría de calculadores. La operación de ordenado debe efectuarse antes de la ejecución del programa que ha de utilizar los datos en cuestión.

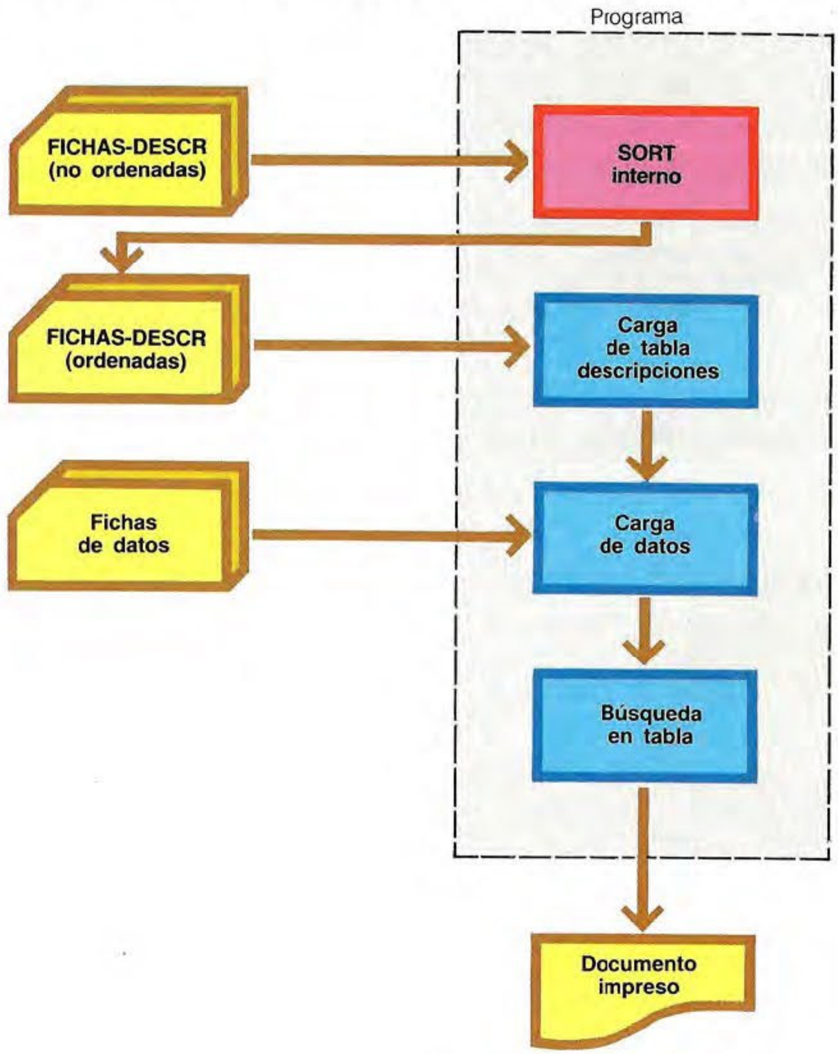
Para fijar ideas, se hará referencia al file de fichas FICHAS-DESCR del ejemplo, que contiene las descripciones de los artículos.
Adoptando una selección exterior, el diagrama de flujo de los datos puede observarse en la página siguiente.
Como alternativa puede ordenarse el file de datos según la clave considerada recurriendo a la selección (Sort) interna de los datos (si la tiene el Compilador).
Este tipo de selección se describirá ampliamente en el capítulo siguiente, pero puede anticiparse que, en el caso de que se adopte esta solución, el flujo de los datos es el esquematizado en la figura de la pág. 1124.
Como última solución puede ordenarse la tabla

directamente en la memoria, inmediatamente después de su carga, mediante cualquier algoritmo gestionado por el programador.
En este caso, el diagrama de flujo asume el aspecto de la figura de la página 1125.
Los tres métodos descritos presentan ventajas y desventajas en términos de tiempo empleado y de recursos de proceso requeridos (memoria central y memoria masiva).
Independientemente del método elegido, es lícito preguntarse: «¿Está justificado penalizar el programa con el tiempo empleado para seleccionar los datos en el intento de usar la SEARCH ALL, o conviene adoptar una búsqueda más lenta sin estar obligados a ordenar los datos de entrada?»
No existe una respuesta precisa a esta pregunta, puesto que la elección depende del tipo de proceso requerido.
Es evidente que si en el curso de un programa pocas veces se tiene necesidad de consultar una tabla, aunque sea grande, pocas veces convendrá utilizar una búsqueda secuencial, evitando el oneroso ordenado de los records del file de entrada o de los elementos de la tabla. Y viceversa, si el proceso requiere el acceso a la tabla muy a menudo, por ejemplo para controlar la validez de todos los records leídos por un file

BUSQUEDA DICOTOMICA PREVIO ORDENADO EXTERNO DE LOS DATOS



BUSQUEDA DICOTOMICA PREVIA ORDENACION INTERNA DE LOS DATOS



de entrada, la búsqueda secuencial resulta extremadamente pesada y, por tanto, es oportuno recurrir a la SEARCH ALL.

Si el calculador utilizado no dispone de un programa para el ordenado de los datos (Sort), o si el Compilador Cobol instalado no tiene las funciones de Sort interno, el único medio que le queda al programador para seleccionar los datos de la tabla es realizar él mismo el ordenado deseado.

En el campo de los microordenadores y ordenadores personales, el tema de la ordenación de una tabla en la memoria es el más tratado, y para ello existen muchísimos algoritmos más o menos rápidos.

Ordenado de los datos (Sort)

Uno de los instrumentos más potentes de que dispone el Cobol es el Sort interno, o bien la posibilidad de seleccionar los records de un file secuencial en orden creciente o decreciente con respecto a unos determinados campos clave. El ordenado puede producirse sobre varias claves y de manera diferente.

Para aclarar esto, consideremos el caso de un file de fichas que tiene la siguiente descripción:

```
01 SK.
   05 LOCALIDAD      PIC X(20).
   05 CABEZA-PART    PIC X(20).
```

```
05 REGION          PIC X(20).
05 NUM-HABIT       PIC 9(8).
05 SITUACION       PIC X(5).
05 FILLER          PIC X(8).
```

En cada ficha hay indicados:

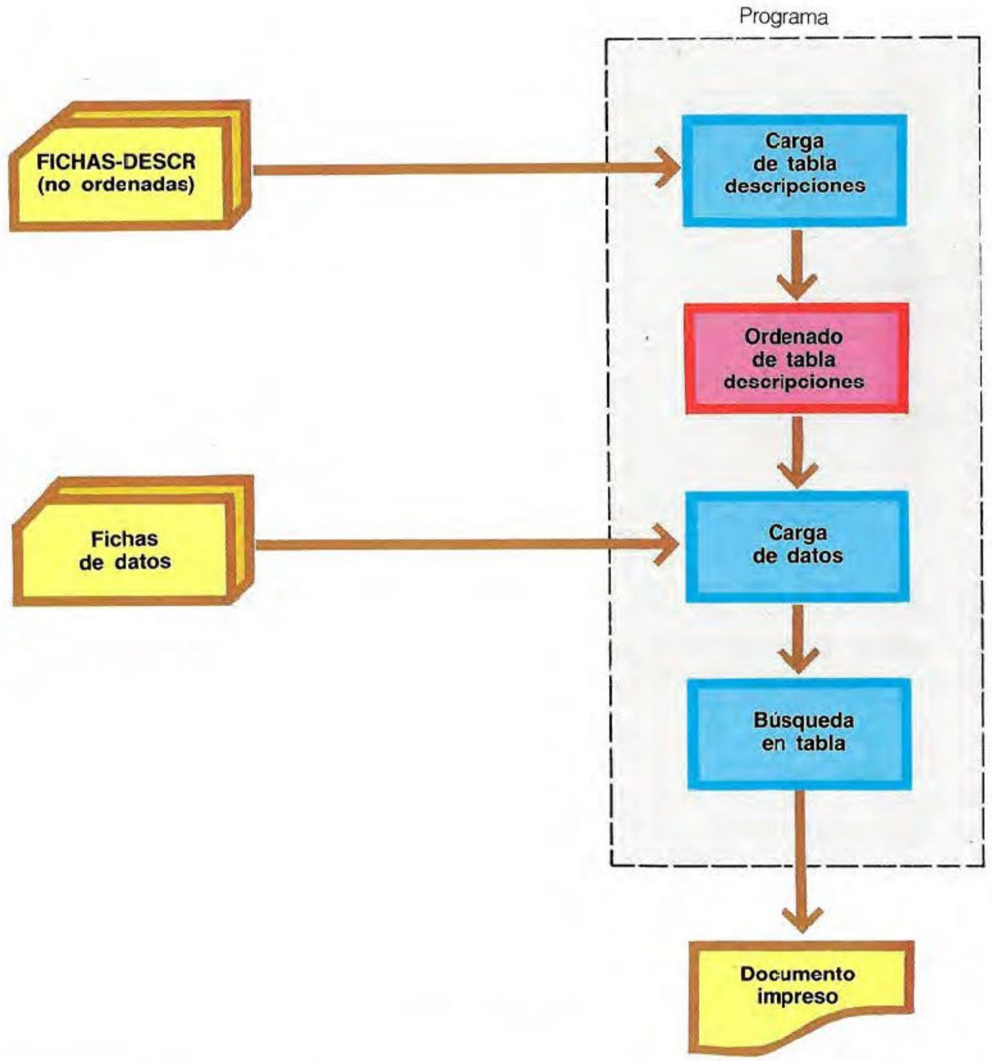
1 / el nombre de la localidad
2 / la cabeza de partido a la que pertenece
3 / la región a la que pertenece
4 / el número de habitantes de la localidad
5 / la situación (Norte o Sur) en que se encuentra con respecto a Madrid.

Queremos seleccionar las fichas para que

- las regiones estén ordenadas de modo ascendente de nombre
- todas las cabezas de partido de una misma provincia estén también ordenadas en secuencia ascendente
- todas las localidades de una misma provincia estén ordenadas por número decreciente de habitantes

La compleja selección requerida puede obte-

BUSQUEDA DICOTOMICA CON ORDENADO GESTIONADO POR EL PROGRAMADOR



EJEMPLO DE ORDENADO Y SELECCION DE DATOS

```
IDENTIFICACION DIVISION.
 .
 .
 .
ENVIRONMENT DIVISION.
 .
 .
FILE-CONTROL.
         SELECT FILE-DE-SORT  ASSIGN TO DISC.
         SELECT FICHAS        ASSIGN TO CARD-READER.
         SELECT FICHAS-ORD    ASSIGN TO DISC.
*
*
 DATA DIVISION.
 FILE SECTION.
 FD  FICHAS
             LABEL RECORD OMITTED.
 01  SK                                PIC X(80).
 FD  FICHAS-ORD
             LABEL RECORD STANDARD.
 01  SK-ORD                            PIC X(30).
 SD  FILE-DE-SORT.
 01  SK-SORT.
     05  LOCALIDAD                     PIC X(20).
     05  CABEZA-PART                   PIC X(20).
     05  REGION                        PIC X(20).
     05  NUM-HABIT                     PIC 9(8).
     05  SITUACION                     PIC X(5).
     05  FILLER                        PIC X(8).
*
*
*
 PROCEDURE DIVISION.
 SELECCIONA.
     SORT FILE-DE-SORT
                       ON ASCENDING KEY REGION




                                         CABEZA-PART
                       ON DESCENDING KEY NUM-HABIT
            USING FICHAS
            GIVING FICHAS-ORD.
*
*
 FIN.
     STOP RUN.
```

nerse con una sola instrucción:

```
SORT file-de-sort
        ON ASCENDING KEY REGION
                             CABEZA-PART
        ON DESCENDING KEY NUM-HABIT
        USING file-a-seleccionar
        GIVING file-seleccionado.
```

Del ejemplo indicado pueden extraerse algunas indicaciones: la función SORT tiene necesidad de conocer

— qué file debe someterse a selección (USING file-a-seleccionar)
— en qué file debe depositar los datos seleccionados (GIVING file-seleccionado)
— en qué área de trabajo debe efectuar la selección (file-de-Sort)
— según qué campos del record y en qué orden debe efectuar la selección (ON ASCENDING KEY...)

Para comprender mejor las posibilidades de la instrucción SORT, a continuación analizaremos las diversas entidades interesadas.

File de Sort. Esta área puede asignarse tanto a DISC como a TAPE, utilizando la cláusula normal SELECT en ENVIRONMENT DIVISION.
Las particularidades más notables para la definición del file de Sort corresponden a la DATA DIVISION, donde

1 / **debe** utilizarse el indicador particular de nivel SD (Sort Definition) en lugar del conocido FD
2 / **no debe** utilizarse la cláusula LABEL RECORD.

Hecha la salvedad de las dos excepciones indicadas, permanecen válidas todas las demás cláusulas vistas para la descripción de los files.
La descripción a nivel SD del file de Sort no define un file físico, sino que permite al Compilador identificar los campos del record según cómo debe hacerse la selección: consigue que no sea necesario describir todos los campos del record a tratar si éstos no participan en la selección. Por otra parte, es aconsejable describir sólo los campos estrechamente interesados en el proceso de selección, que de este modo se obtienen procesados más ágiles.

File a seleccionar. Es un file secuencial cualquiera utilizado en INPUT, pero que no necesita apertura (OPEN) ni cierre (CLOSE) por parte del programador cuando se utiliza la cláusula USING.

File seleccionado. También es un file secuencial que subyace en las operaciones normales de definición de los files. Está completamente gestionado por el Compilador en apertura y en cierre cuando se usa la cláusula GIVING.

Orden de selección. Cada preposición ON encontrada en la instrucción SORT especifica el tipo de ordenado que debe aplicarse a los campos clave relacionados.
La secuencia en que se disponen estas claves en el interior de la instrucción determina el nivel jerárquico de importancia de las propias claves. En el ejemplo indicado, la clave REGION resulta de mayor importancia que la clave CABEZA-PART, y análogamente, ésta es prioritaria con respecto a NUM-HABIT.
El listado de este programa de selección de fichas se ha representado en la página anterior.

La INPUT PROCEDURE y el verbo RELEASE

En general, la función de Sort es muy onerosa en términos de proceso. Por tanto, es una buena norma seleccionar sólo los records indispensables en el proceso, descartando los demás.
Para permitir que el programador pueda elegir los records a «incluir en el Sort» en base a condiciones establecidas, el Cobol prevé una INPUT PROCEDURE alternativa a la cláusula USING.
En esta SECTION, el record leído por el file de entrada se somete a procesos o controles antes de ser descartado o incluido en el Sort.
Esta última operación consiste en escribir el record en el file de Sort mediante una adecuada instrucción, cuyo formato se ha representado en la tabla de abajo.
Como puede observarse, la instrucción es completamente análoga a la WRITE. Para ilustrar la función introducida, supongamos que deben seleccionarse, según las especificaciones indicadas, sólo las fichas correspondientes a las localidades con un número de habitantes superior a los 20.000.
Omitiendo las partes comunes al ejemplo anterior, la nueva estructura del programa se ha indicado en el listado de la página siguiente.

La OUTPUT PROCEDURE y el verbo RETURN

En lugar de la cláusula GIVING, el Cobol permite declarar al Compilador el nombre de una SECTION (OUTPUT PROCEDURE), en la que el programador puede someter a cualquier tipo de proceso los records seleccionados y leídos directamente por el file de Sort.
Análogamente a lo que sucede con la INPUT PROCEDURE, el acceso al file de Sort no puede hacerse con el verbo READ, sino sólo con la adecuada instrucción RETURN, cuyo formato general se ha representado en la tabla de la página siguiente.

FORMATO DE LA INSTRUCCION RELEASE

RELEASE record-de-sort [FROM nombre-de-dato]

EJEMPLO DE USO DE LA INSTRUCCION RELEASE

```
 WORKING-STORAGE SECTION.
 01  SK-WS.
     05  LOCALIDAD-WS              PIC X(20).
     05  CABEZA-PART-WS            PIC X(20).
     05  REGION-WS                 PIC X(20).
     05  NUM-HABIT-WS              PIC 9(8).
     05  SITUACION-WS              PIC X(5).
     05  FILLER                    PIC X(8).
*
*
*
 PROCEDURE DIVISION.
 SELECCIONA SECTION.
 INICIO.
     OPEN INPUT-FICHAS.
     SORT FILE DE SORT
                        ON ASCENDING KEY REGION
                                         CABEZA-PART
                        ON DESCENDING KEY NUM-HABIT
           INPUT PROCEDURE LIBERA
           GIVING FICHAS-ORD.
 FIN.
     STOP RUN.
*
*
*
 LIBERA SECTION.
 LEE-FICHAS.
     READ FICHAS INTO SK-WS
                 AT END
                 CLOSE FICHAS
                 GO TO LIBERA-EX.
     IF NUM-HABIT-WS GREATER THAN 20000
        RELEASE SK-SORT FROM SK-WS.




     GO TO LEE-FICHAS.
 LIBERA-EX.
     EXIT.
```

FORMATO GENERAL DE LA INSTRUCCION RETURN

<u>RETURN</u> file-de-sort [<u>INTO</u> nombre-de-dato]

<u>AT END</u> frase-imperativa

Siempre haciendo referencia al mismo ejemplo, supongamos que deben memorizarse en dos files diferentes las localidades situadas, respectivamente, al Norte y al sur de Madrid.
El detalle del programa se ha representado en el listado de las págs. 1130 y 1131.

Gestión de los files de índices

Todos los files que hemos considerado hasta ahora tienen una organización de tipo secuencial. Seguidamente nos ocuparemos de los files de organización en índices. A diferencia de los files secuenciales, en los que siempre deben leerse todos los records que preceden al deseado, en el caso de los files de índice basta con indicar en el campo clave del record el valor buscado para obtener rápidamente dicho record. La organización en índices de un file permite así acceder directamente a un record a través de una clave simbólica.
Por ejemplo, supongamos que deben buscarse algunas informaciones relativas a un empleado del que se conoce la matrícula.
Las informaciones relativas a todos los empleados se han registrado en un file ordenado por número de matrícula. Si el file tuviese una organización secuencial, debería utilizarse el procedimiento ilustrado en el diagrama de flujo de la pág. 1131, o sea, deberían leerse todos los records del file hasta encontrar el correspondiente a la matrícula a examinar.
Como ya se ha indicado, sería necesario realizar como promedio un número de accesos igual a la mitad del número de records en el file.
En cambio, si el file tiene una organización de índices, en la que la matrícula se ha definido como clave de acceso, basta con implantar en la clave la matrícula deseada y efectuar la lectura: el record queda disponible rápidamente sin necesidad de efectuar bucles de lectura.
El diagrama de flujo de la pág. 1132 ilustra el procedimiento.
El tipo de búsqueda descrito es posible gracias a una estructura particular del file que, además de los records de datos, contiene una tabla **(tabla de las direcciones)** en la que se han memorizado todas las claves con la dirección del record a que hace referencia cada clave. La estructura se ha representado esquemáticamente

Un momento de la comprobación de una impresora.

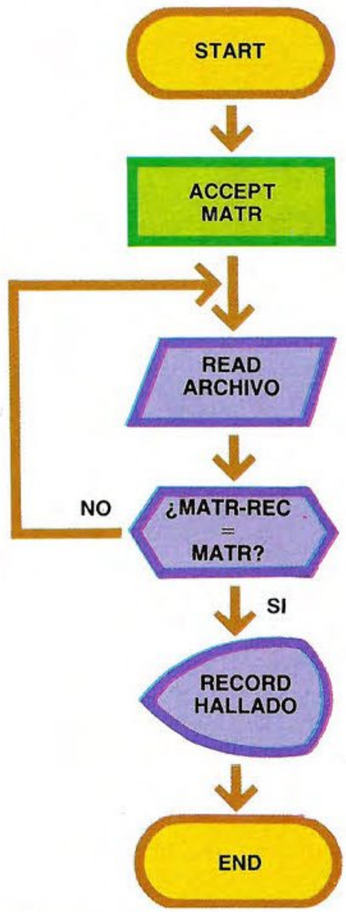


EJEMPLO DE USO DE LA INSTRUCCION RETURN

```
IDENTIFICATION DIVISION.
 .
 .
ENVIRONMENT DIVISION.
 .
 .
FILE-CONTROL.
     SELECT FILE-DE-SORT    ASSIGN TO DISC.
     SELECT FICHAS          ASSIGN TO CARD-READER.
     SELECT FICHAS-NORTE    ASSIGN TO DISC.
     SELECT FICHAS-SUR      ASSIGN TO DISC.
*
DATA DIVISION.
FILE SECTION.
FD  FICHAS
          LABEL RECORD OMITTED.
01  SK                              PIC X(80).
FD  FICHAS-NORTE
          LABEL RECORD STANDARD.
01  SK-NORTE                        PIC X(80).
FD  FICHAS-SUR
          LABEL RECORD STANDARD.
01  SK-SUR                          PIC X(80).
SD  FILE-DE-SORT.
01  SK-SORT.
    05  LOCALIDAD                   PIC X(20).
    05  CABEZA-PART                 PIC X(20).
    05  REGION                      PIC X(20).
    05  NUM-HABIT                   PIC 9(8).
    05  SITUACION                   PIC X(5).
    05  FILLER                      PIC X(8).
*
*
*
WORKING-STORAGE SECTION.
01  SK-WS.
    05  LOCALIDAD-WS                PIC X(20).
    05  CABEZA-PART-WS              PIC X(20).
    05  REGION-WS                   PIC X(20).
    05  NUM-HABIT-WS                PIC 9(8).
    05  SITUACION-WS                PIC X(5).
    05  FILLER                      PIC X(8).
*
*
*
PROCEDURE DIVISION.
SELECCIONA SECTION.
INICIO.
    OPEN INPUT FICHAS.
    SORT FILE-DE-SORT
                        ON ASCENDING KEY REGION
                                         CABEZA-PART
                        ON DESCENDING KEY NUM-HABIT
          INPUT  PROCEDURE  LIBERA
          OUTPUT PROCEDURE  SEPARA.
FIN.
  STOP RUN
*
*
*                    SECCIONES
*
*
LIBERA SECTION.
LEE-FICHAS.
   READ FICHAS INTO SK-WS
                AT END
                CLOSE FICHAS
                GO TO LIBERA-EX.
```

```
     IF   NUM-HABIT-WS GREATER THAN 20000
          RELEASE SK-SORT FROM SK-WS.
     GO TO LEE-FICHAS.
 LIBERA-EX.
     EXIT.
*
*
*
 SEPARA SECTION.
 ABRE-FILES.
     OPEN OUTPUT FICHAS-NORTE
                 FICHAS-SUR.
 LEE-SORT.
     RETURN FILE-DE-SORT INTO SK-WS
                         AT END
                         CLOSE FICHAS-NORTE
                         CLOSE FICHAS-SUR
                         GO TO SEPARA-EX.
     IF   SITUACION-WS = 'NORTE'
          WRITE SK-NORTE FROM SK-WS.
     IF   SITUACION-WS = 'SUR'
          WRITE SK-SUR FROM SK-WS.
     GO TO LEE-SORT.
 SEPARA-EX.
     EXIT.
```

EJEMPLO DE BUSQUEDA EN UN FILE SECUENCIAL



EJEMPLO DE BUSQUEDA EN UN FILE DE INDICES



en la tabla de arriba de la página siguiente. Cuando se pide la lectura de un record especificando una clave, el programa efectúa una o más lecturas en la tabla de las direcciones hasta encontrar la clave buscada y, a continuación, utilizando la dirección definida en la tabla, accede directamente al record. Por tanto, para leer un record no se hace una sola lectura, sino dos, independientemente de la posición del record. Esto resulta particularmente conveniente cuando es necesario efectuar la búsqueda casual en un file de dimensiones bastante grandes.

Las modalidades según las que es posible acceder a un file de índices son tres:

1 / casual (RANDOM)
2 / secuencial (SEQUENTIAL)
3 / mixta o dinámica (DYNAMIC)

Adoptando la modalidad de acceso SEQUENTIAL, el file queda sometido a las reglas, ya indicadas, válidas para el tratamiento de los files secuenciales: los records se procesan en secuencia física, procediendo del primero al último.

Las particularidades del acceso RANDOM también se han explicado.

La tercera modalidad de acceso (DYNAMIC) es una síntesis de las otras dos y permite un tratamiento mixto del file, en el sentido de que es posible adoptar la modalidad más oportuna a las necesidades del momento. Por ejemplo, es posible leer una serie de records secuencialmente, leer otros en forma RANDOM, volver a usar el acceso SEQUENTIAL, y así sucesivamente.

Gracias a las características descritas, un file de índices también se llama **IS** (INDEXED-

ESQUEMA DE LA ESTRUCTURA DE UN FILE DE INDICES

Tabla de las direcciones

Clave (matrícula)	Dirección del record
0 0 0 1 5	0 0 0 0 0 1
0 0 0 3 8	0 0 0 0 0 2
0 0 0 8 8	0 0 0 0 0 3
0 0 1 3 3	0 0 0 0 0 4
0 0 1 3 4	0 0 0 0 0 5

Record de datos

Matrícula	Apellido y nombre
0 0 0 1 5	ROJO CARLOS
0 0 0 3 8	LUJAN MARIO
0 0 0 8 8	BRU LUCIA
0 0 1 3 3	MAESTRO JUAN
0 0 1 3 4	JUEZ RAMON

SEQUENTIAL) o **secuencial de índices**. A continuación ilustraremos las cláusulas para la definición de un file IS en ENVIRONMENT DIVISION y en DATA DIVISION, así como las intrucciones de la PROCEDURE DIVISION, necesarias para su gestión.

Descripción de los files IS en ENVIRONMENT DIVISION

Si se tiene la intención de utilizar en un programa un file secuencial de índices (IS), a nivel SELECT, el file debe asignarse necesariamente a DISC y debe especificarse la cláusula

ORGANIZATION IS INDEXED

A ésta debe seguirla una segunda cláusula, necesaria para la declaración de la modalidad de acceso, o sea

ACCESS MODE IS { SEQUENTIAL | RANDOM | DYNAMIC }

Finalmente debe haber la cláusula

RECORD KEY IS nombre-de-dato

«nombre-de-dato» es el nombre de un campo del record, definido en la File Description, que contiene la clave simbólica del propio record. Este campo puede ser elemental o de grupo, y generalmente no puede tener una longitud superior a 60 caracteres.
El campo clave no puede contener un valor equivalente a LOW-VALUES, y debe implantarse antes de cada operación de acceso casual.

Descripción de los files IS en DATA DIVISION

En la File Description es necesario utilizar la cláusula

LABEL RECORD STANDARD

según las normas definidas anteriormente para los files secuenciales asignados a DISC.
En la descripción del record, siempre a nivel FD, es necesario definir el campo que contiene la clave del record con el nombre indicado en la cláusula RECORD KEY a nivel SELECT.

Uso de las instrucciones en la PROCEDURE DIVISION

Un file de índices puede abrirse en los tres modos ya vistos para los files de acceso secuencial, con exclusión de la modalidad EXTEND.
Por tanto, el formato de la instrucción de apertura es el siguiente:

FORMA DE LA INSTRUCCION OPEN

OPEN { INPUT | OUTPUT | I-O } nombre-de-file

El verbo READ. La instrucción de lectura puede utilizarse sólo si el file se ha abierto previamente en INPUT o en I-O, y tiene dos formatos según la modalidad de acceso definida a nivel SELECT. Los dos formatos se han ilustrado en las dos tablas de arriba de la página siguiente. Más precisamente, valen estas reglas:

PRIMER FORMATO DE LA INSTRUCCION READ APLICADA A FILES IS

READ nombre-de-file [NEXT] RECORD [INTO nombre-de-dato-1]
AT END frase-imperativa.

SEGUNDO FORMATO DE LA INSTRUCCION READ APLICADA A FILES IS

READ nombre-de-file [INTO nombre-de dato-2]
INVALID KEY frase-imperativa.

1 / En todos los files para los cuales se ha definido el acceso secuencial (ACCESS MODE IS SEQUENTIAL) debe usarse el primer formato sin la cláusula NEXT, o sea

READ nombre-de-file RECORD
[INTO nombre-de-dato-1]
AT END frase-imperativa.

2 / En lecturas de tipo secuencial en files definidos con acceso mixto (ACCESS MODE IS DYNAMIC) debe emplearse el primer formato con la cláusual NEXT, o sea

READ nombre-de-file NEXT RECORD
[INTO nombre-de-dato-1]
AT END frase-imperativa.

3 / En todas las operaciones de lectura casual, cuando a nivel SELECT se ha declarado ACCESS MODE IS RANDOM, o bien ACCESS MODE IS DYNAMIC, debe usarse el segundo formato.

4 / Antes de la operación de lectura, el campo de la RECORD KEY debe implantarse al valor correspondiente al del record que se está buscando. Si el record con la clave antes especificada no existe, se ejecutarán las instrucciones de la «frase-imperativa» que sigue a la cláusula INVALID KEY.

El verbo WRITE. Esta instrucción puede utilizarse sólo en files previamente abiertos en OUTPUT o I-O, y tiene el formato indicado en la tabla al final de esta página.

En el caso en que el file se abre en OUTPUT (fase de creación), los records deben escribirse en orden ascendente de claves. Es decir, se accede al file de manera secuencial cualquiera que sea el ACCESS MODE especificado.

Antes de efectuar cada escritura debe implantarse el campo del record que contiene la RECORD KEY.

La frase imperativa especificada en la cláusula INVALID KEY solamente se ejecuta en los siguientes casos:

- cuando los valores de las claves no están en orden ascendente de una WRITE a la siguiente
- cuando el valor de la clave no es único en el ámbito del file
- cuando se ha llenado todo el espacio ubicado en el disco.

Instrucciones específicas para la gestión de los files IS

Hay tres verbos que no tienen equivalentes en los files de tipo secuencial:

REWRITE se usa para la actualización de records existentes

FORMATO DE LA INSTRUCCION WRITE APLICADA A FILES IS

WRITE nombre-de-record [FROM nombre-de-dato]
INVALID KEY frase-imperativa.

DELETE cancela los records ya presentes en el file

START permite posicionarse en un determinado record

El verbo REWRITE. La operación de reescritura (REWRITE) sólo se permite en files abiertos en I-O y tiene el formato indicado en la primera tabla de abajo.

El uso de la instrucción REWRITE está estrechamente vinculado al respeto de algunas reglas fundamentales:

— la instrucción sólo puede ejecutarse si al modificar el record no queda modificaca la clave de acceso.
  Si esto sucediese, se ejecutaría la frase imperativa que se ha especificado en la cláusula INVALID KEY
— en el caso de files definidos con ACCESS MODE SEQUENTIAL es necesario ejecutar con éxito una READ antes de la instrucción REWRITE
— la frase imperativa especificada en la cláusula INVALID KEY sólo se ejecuta si el valor de la clave implantada en la RECORD KEY antes de la REWRITE es diferente del que hay en cualquier record memorizado en el file. O sea, no es posible efectuar la reescritura de un record que todavía no se ha creado.

El verbo DELETE. La instrucción de cancelación (DELETE) tiene el formato representado en la segunda tabla de abajo. Análogamente a la instrucción de reescritura (REWRITE), la DELETE debe trabajar sobre records ya existentes. Las reglas que la rigen, por tanto, son similares a las descritas para la instrucción REWRITE:

— el file tiene que haber sido abierto en I-O
— si el file tiene ACCESS MODE SEQUENTIAL, deberá ejecutarse con éxito una READ antes de la DELETE
— si el file se ha definido con acceso RANDOM o DYNAMIC, basta con implantar la RECORD KEY antes de efectuar la cancelación, sin tener que leer previamente el record que debe cancelarse
— si a la clave implantada no corresponde ningún record, se ejecuta la frase imperativa especificada en la cláusula INVALID KEY.



Fase de comprobación de terminales vídeo.

FORMATO DE LA INSTRUCCION REWRITE

REWRITE nombre-de-record [FROM nombre-de-dato]
INVALID KEY frase-imperativa.

FORMATO DE LA INSTRUCCION DELETE

DELETE nombre-de-file RECORD
INVALID KEY frase-imperativa.

El verbo START. La instrucción START permite posicionarse sobre un determinado record del file, ya colocado, y leer los siguientes de forma secuencial.
Por ejemplo, si hubiese que imprimir todos los records del file de los empleados definido anteriormente, en el que la matrícula, o sea RECORD KEY, es superior a un cierto valor, es posible posicionarse en la primera matrícula válida mediante el verbo START y después pueden imprimirse todos los records que quedan leyéndolos secuencialmente del file. En ésta y en la página siguiente se han representado para su comparación los diagramas de flujo de la operación en el caso de que se recurra a la START o a otro método alternativo.
Obsérvese que la instrucción START no deja disponible efectivamente el record, sino que sólo hace un apuntado a la posición deseada.
La instrucción START debe ir seguida de una

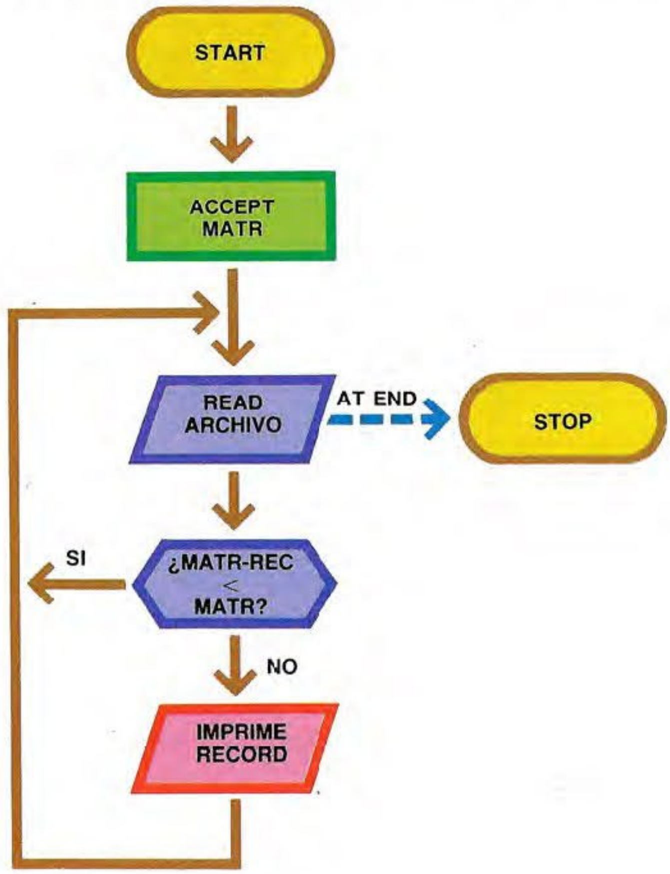
READ nombre-de-file AT END...

o de una

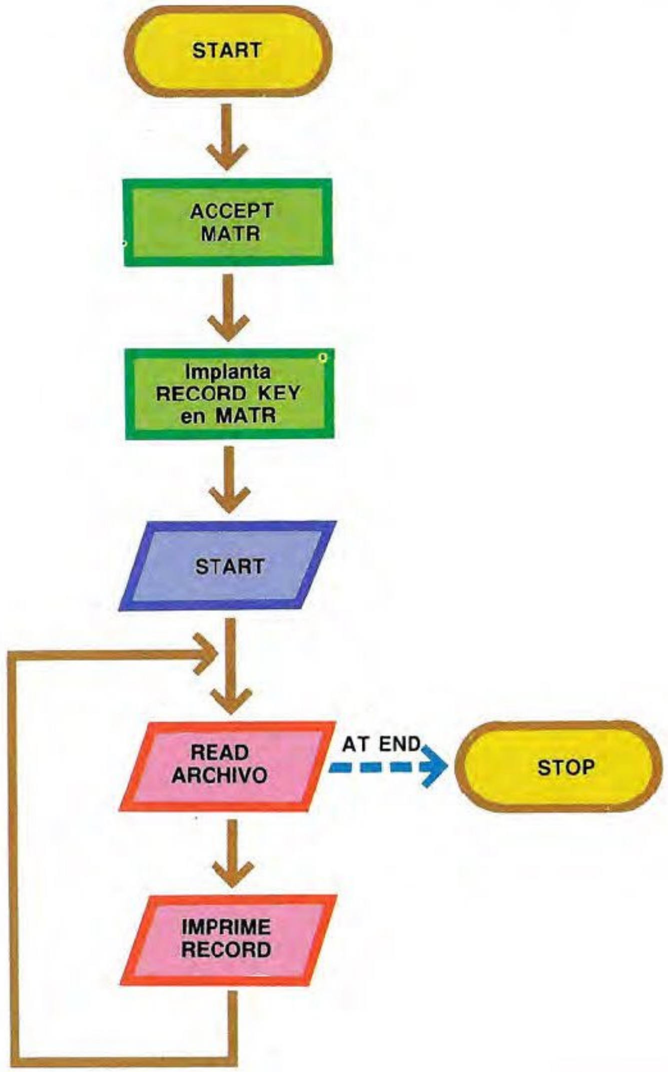
READ nombre-de-file NEXT AT END...

según si el file se ha declarado con acceso secuencial o dinámico respectivamente. La operación de posicionado sólo puede hacerse sobre files abiertos en INPUT o en I-O, y cuando el acceso se haya definido como SEQUENTIAL o DINAMIC.
El formato general de la instrucción se ha representado en la tabla de la página siguiente.
El "nombre-de-dato" debe ser el nombre de la RECORD KEY y debe implantarse al valor deseado antes de la ejecución de la START.
En el caso de que esté omitida la cláusula KEY, el Compilador supondrá que el record buscado tiene la clave igual a la implantada en la RE-

POSICIONADO EN UN RECORD DEL FILE IS SIN START



POSICIONADO EN UN RECORD DEL FILE IS CON START



FORMATO GENERAL DE LA INSTRUCCION START

START nombre-de-file KEY is { EQUAL TO / = / GREATER THAN / > / NOT LESS THAN / NOT < } nombre-de-dato

INVALID KEY frase-imperativa.

CORD KEY. Es decir, omitir la cláusula KEY equivale a especificarla en la forma KEY IS EQUAL TO.
Si el record buscado no existe se ejecuta la frase especificada en la cláusula INVALID KEY.
Cuando la cláusula KEY se especifica en la forma GREATER THAN o NOT LESS THAN, el sistema se posiciona en el primer record cuya clave tiene un valor (respectivamente) mayor o no menor que el implantado en la RECORD KEY antes de la operación START.
En estos casos, la frase imperativa de la cláusula INVALID KEY sólo se ejecuta si la clave implantada en la RECORD KEY es mayor que la clave más alta memorizada en el file.

El verbo CLOSE. La operación de cierre de un file IS es perfectamente análoga, en el formato y en el uso, a la utilizada para los files de tipo secuencial, ya ilustrada:

CLOSE nombre-de-file

La tabla de abajo contiene una recopilación de las operaciones ejecutables en files de índices. Para cada operación se ha indicado en qué condiciones de apertura o de acceso está permitido su uso.
En cambio, en la tabla de la página siguiente se ha representado la lista de las causas de salida por INVALID KEY frente a la operación realizada y en función de las modalidades de apertura y de acceso utilizadas.

Comunicación entre programas. Las subrutinas

Un programa en formato ejecutable puede estar compuesto por un conjunto de subprogramas (subrutinas) unidos lógica y físicamente entre sí por un programa principal (main).
La ventaja de este tipo de organización reside en que cada subprograma sencillo puede compilarse y probarse de manera completamente autónoma con respecto a los otros, y esto permite reducir de forma considerable los tiempos de escritura y de prueba.
Otra ventaja que se deriva del uso de las subrutinas es que con esta técnica se crea automáticamente una librería de funciones de proceso asimismo complejas que pueden utilizarse también para programas diferentes que el que ha originado la primera necesidad: piénsese por ejemplo en un subprograma que controla la validez de una fecha, o bien que determina el día de la semana que corresponde a la fecha.
Estos dos subprogramas de uso general se re-

RECOPILACION DE LAS OPERACIONES REALIZABLES EN FILES IS

OPERACION	OPEN			ACCESS MODE		
	INPUT	OUTPUT	I-O	SEQ	DYN	RAN
OPEN INPUT ...	*			*	*	*
OPEN OUTPUT ...		*		*		
OPE I-O ...			*	*	*	*
READ .... AT END	*		*	*		
READ .... NEXT AT END ...	*		*		*	
READ .... INVALID KEY ...	*		*			*
WRITE ... INVALID KEY ...		*		*		
WRITE ... INVALID KEY ...			*	*	*	*
REWRITE . INVALID KEY ...			*	*	*	*
DELETE			*	*		
DELETE .. INVALID KEY ...			*		*	*
START ... INVALID KEY ...	*		*	*	*	

SEQ = SEQUENTIAL DYN = DYNAMIC RAN = RANDOM

CAUSAS DE SALIDA POR INVALID KEY DE UN FILE IS

OPERACION	OPEN	ACCESS MODE	INVALID KEY
READ	I/I-O	RAN	No existe un record con clave igual a la implantada.
WRITE	O	SEQ	Falta espacio en disco.
WRITE	I-O	—	Ya existe un record con clave igual. Falta espacio en disco.
REWRITE	I-O	SEQ	La clave no corresponde a la del record leído anteriormente.
REWRITE	I-O	DYN/RAN	No existe un record con clave igual a la implantada.
DELETE	I-O	DYN/RAN	No existe un record con clave igual a la implantada.
START	I-O	SEQ/DYN	No existe un record con clave igual a la implantada.
START ... KEY EQUAL ...	I-O	SEQ/DYN	No existe un record con clave igual a la implantada.
START ... KEY GREATER ...	I-O	SEQ/DYN	No existe un record mayor que la clave que se ha implantado.
START ... KEY NOT LESS	I-O	SEQ/DYN	No existe un record con clave mayor o igual a la que se ha implantado.

SEQ = SEQUENTIAL DYN = DYNAMIC RAN = RANDOM
I = INPUT O = OUTPUT I-O = INPUT-OUTPUT

presentarán como ejemplos de aplicación al final del capítulo.

Es interesante observar que el programa principal y sus subprogramas están unidos físicamente durante la operación de encadenado, por lo que la ocupación de memoria real del programa compuesto viene dada por la suma de las ocupaciones de los programas que lo componen.

Cuando un programa, ya sea el principal o una subrutina, reclama un subprograma para realizar una determinada función, generalmente debe comunicar a este último los datos que van a utilizarse.

Si por ejemplo se considera una subrutina para el control de la fecha, el programa llamador deberá proporcionar la fecha en la que el programa lamado debe efectuar los controles.

Por otra parte, el subprograma deberá comunicar al programa principal la salida del control efectuado.

Este intercambio de informaciones se obtiene utilizando campos de memoria identificados con nombres simbólicos que deben describirse a nivel 01 o 77 de la WORKING-STORAGE SECTION del programa llamador.

Los correspondientes campos del subprograma deben estar descritos, según las reglas conocidas, en una sección adecuada de la DATA DIVISION.

Esta sección, denominada LINKAGE SECTION, debe definirse al final de la WORKING-STORAGE SECTION.

En la LINKAGE SECTION no se admite la cláusula VALUE si no corresponde a los niveles 88.

Los nombres de los campos de tránsito no deben ser necesariamente iguales en el programa llamador y el llamado, aunque han de mantener las mismas PICTURE y USAGE a nivel de conjunto.

Las previsiones del ordenador

Hace más de un decenio que los meteorólogos se valen de la ayuda de los satélites. Los datos transmitidos a tierra desde éstos no sólo sirven para el progreso de la meteorología como ciencia teórico-experimental, sino que permiten sobre todo las formulaciones de previsiones del tiempo siempre más fiables, con niveles de detalle que hace pocos años habrían parecido inalcanzables incluso a los más optimistas.

Más allá de la innegable importancia de los satélites, los resultados que hoy hacen de la meteorología una ciencia en rapidísimo progreso se deben en gran parte al ordenador y a las técnicas de proceso automático de los datos.

En Italia se ha constituido un nuevo centro para la recepción, proceso y presentación de datos obtenidos por satélites meteorológicos en el Centro Nazionale di Meteorologia e Climatologia Aeronautica (CNMACA) del Servizio Meteorologico dell'Aeronautica Militare, y tiene por objetivo la utilización de las informaciones del satélite geoestacionario europeo Meteosat.

La red nacional italiana de adquisición y tratamiento de datos está constituida por una estación receptora primaria, por cinco estaciones receptoras secundarias, por un sistema de proceso de datos y por veinte estaciones automáticas de detección.

La estación receptora primaria (PDUS, Primary Data User Station) puede adquirir las informaciones digitales del Meteosat, mientras que las estaciones receptoras secundarias (SDUS, Secondary Data User Station) pueden adquirir las informaciones analógicas del Meteosat. Efectivamente, el satélite envía a tierra dos tipos diferentes de señales: analógicas y digitales.

La señal analógica es una entidad que varía de forma casi continua manteniéndose rigurosamente proporcional a la magnitud detectada. En cambio, la señal digital está constituida por una secuencia de impulsos de tipo si-no, o sea, del mismo tipo que los procesados en los calculadores. En el segundo caso, la proporcionalidad con la magnitud observada se traduce en términos numéricos.

La principal magnitud mantenida en observación por el Meteosat es la «radiancia» de la superficie terrestre, o sea, la cantidad de energía radiada al espacio por la corteza terrestre, las superficies líquidas y los sistemas nubosos.

El subsistema de cálculo, de tipo «off-line», tiene la misión de procesar los datos contenidos en las señales digitales o analógicas y traducirlos en una imagen gráfica.

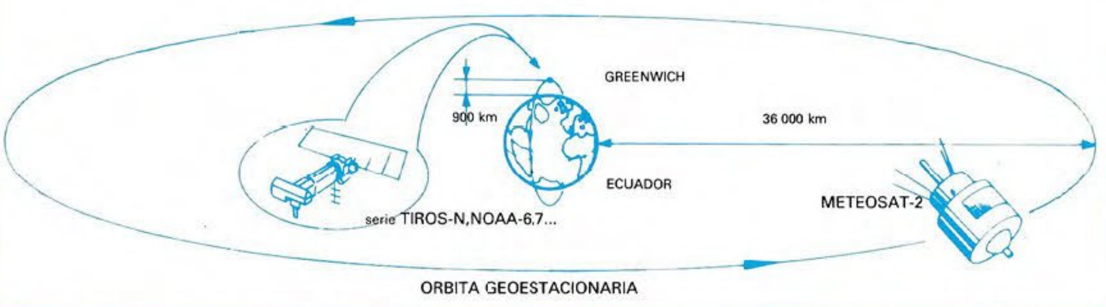
Las veinte estaciones automáticas de recogida de datos del tipo DCP (Data Collection Platform) emplean el Meteosat como puente para la transmisión de dichos datos al Circuito Mundial de Telecomunicaciones Meteorológicas.

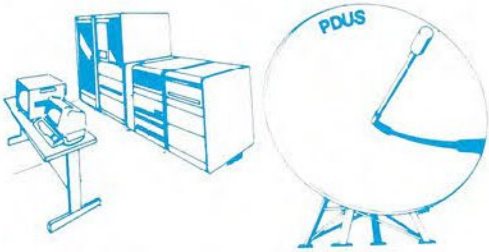
La estación primaria PDUS está constituida por un subsistema de recepción, dotado de una antena parabólica de cerca de 5 metros de diámetro, de un subsistema de cálculo y de sofisticados aparatos de presentación gráfica y alfanumérica de los datos y de las imágenes. La antena recibe directamente del Meteosat 2 las imágenes de alta resolución y, después de un pretratamiento, se reenvían desde Darmstadt (donde tiene su sede el Centro Operativo que gestiona el satélite) al Meteosat.

El corazón de la estación primaria está constituido por un procesador cuya unidad central de proceso (CPU) está provista de 2 Megabytes de memoria RAM, y que se apoya en memorias masivas, en discos magnéticos para un conjunto de cerca de 250 Mbytes.

Dos unidades de cinta magnética permiten el almacenamiento temporal e histórico de los da-

SATELITES POLARES Y GEOESTACIONARIOS

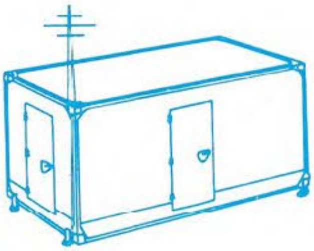




PDUS
PRIMARY DATA USER STATION

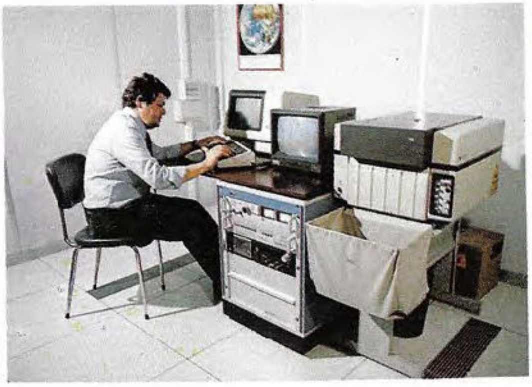


SDUS
SECONDARY DATA USER STATION



DCP
DATA COLLECTION PLATFORM







Arriba, ilustración esquemática de los subsistemas de adquisición y proceso de los datos transmitidos por el satélite Meteosat. A la derecha, arriba, estación de trabajo del sistema ISIDE; en el centro, un momento del trabajo en una estación receptora secundaria (SDUS); abajo, el calculador central de la estación receptora primaria (PDUS) de Roma.

tos, mientras que la gestión operativa de los procedimientos de proceso de datos se hace utilizando una serie de terminales interactivos. Aparatos de reproducción fotográfica digital, conectados al ordenador, permiten también la impresión de partes de imágenes de especial interés meteorológico.

Un conjunto de microordenadores que trabajan en paralelo con la CPU, interfaces y presentadores gráficos permite acumular de manera conti-

nua hasta 32 imágenes de alta resolución para cada una de las tres bandas (visible, infrarrojos y vapor acuoso) en las que trabaja el radiómetro del Meteosat, y realizar películas. Estas últimas se presentan después mediante un sofisticado y eficaz sistema de animación basado en una arquitectura de multiprocesador y provisto de 2 Mbytes de memoria.

Las masas nubosas y las perburbaciones meteorológicas discurren así bajo el ojo del meteorólogo reconstruyendo una «historia» de movimientos y de desarrollo que llega normalmente hasta las 16 horas anteriores a la última imagen, con intervalos de 30 minutos. Ampliando el intervalo de tiempo entre una imagen y la siguiente es posible hacer películas que abarcan un tiempo superior a 16 horas. También pueden realizarse películas de duración muy superior a las 16 horas sobre cinta magnética o videocassette para ulteriores reproducciones.

La posibilidad de empleo del zoom interactivo y de la técnica del falso color, basada en la atribución de un color a determinados valores de radiancia medida por el Meteosat, aumenta notablemente la capacidad de interpretación física de los fenómenos atmosféricos en curso. El color puede emplearse también después de un tratamiento de los datos de radiancia basado en técnicas multiespectrales. Cada una de las 32 imágenes presentadas está constituida por un enorme número de elementos de imagen (pixels): 1250 líneas × 2500 pixels/línea en la banda de luz visible, y 625 × 1250 para la banda de la luz infrarroja o del valor acuoso: para cada pixel, el sistema puede memorizar el valor numérico de la correspondiente radiancia.

El proceso de los datos digitales recogidos mediante la estación primaria permite obtener conjuntos de datos meteorológicosa de gran utilidad para la previsión del tiempo.

Uno de éstos es la medición de la velocidad de los vientos en cota, deducida del desplazamiento de las nubes mediante la observación de varias imágenes sucesivas de la misma zona. La información es de fundamental importancia porque llena el vacío que dejan las observaciones convencionales en cota mediante radiosondas en las zonas desérticas u oceánicas.

La dirección y la velocidad de los vientos deducidas de los datos del Meteosat se entran diariamente como datos en los modelos numéricos de previsión del tiempo a corto y medio plazo.

Una segunda información de gran importancia corresponde a la medición de las temperaturas superficiales de los mares. Éstas pueden obtenerse procesando los datos relativos a la radiancia en la banda del infrarrojo, que es proporcional a la temperatura del cuerpo radiante (ley de Stefan-Boltzmann), teniendo en cuenta algunas correcciones debidas a la atenuación sufrida por la radiación al atravesar la atmósfera y al ángulo de inclinación de la luz solar.

Estos valores superficiales de temperatura de los océanos se utilizan normalmente en los modelos de previsión y para los balances de radiación del sistema Tierra-Atmósfera.

El análisis de las nubes permite determinar, en amplias zonas, el porcentaje de cielo cubierto y las diversas nubes que concurren a formar la nubosidad total. Este tipo de análisis se repite cada 6 horas y es directamente utilizable para la ayuda a la navegación aérea.

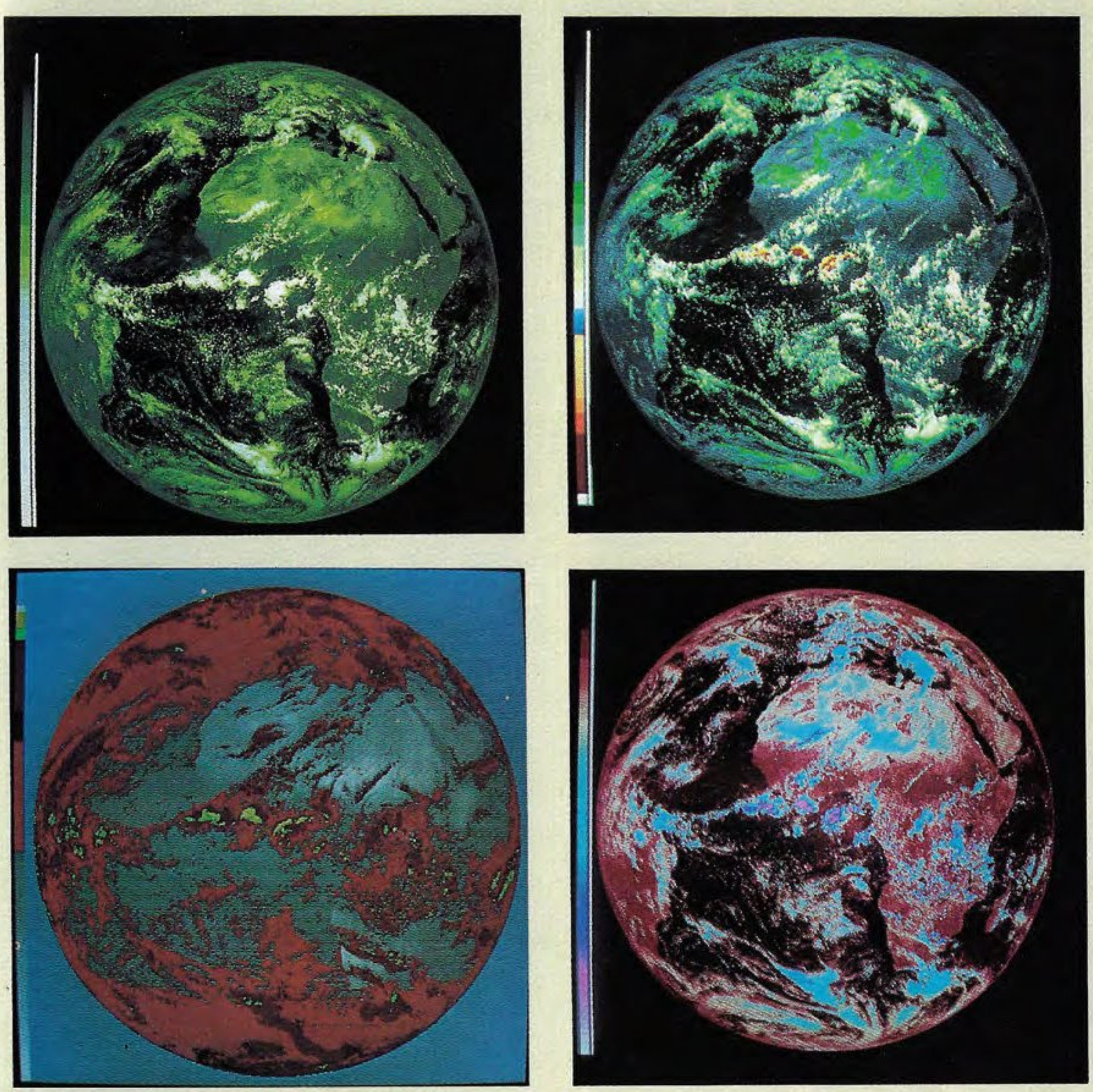
Técnicas estadísticas permiten clasificar los diferentes tipos de nubes y proporcionar además, en condiciones de cielo despejado, útiles informaciones sobre las corrientes marinas.

La cota superior de las nubes es otro dato deducible del análisis de las radiancias en amplias zonas de Europa, sobre el Mediterráneo y de África. Como la temperatura de la cota superior de los cuerpos nubosos es inversamente proporcional a la altitud a que se encuentran, puede aplicarse un color a determinados intervalos de temperatura y presentar en el monitor un mapa representatijvo de la superficie exterior de los sistemas nubosos, así como de su extensión. Ésta es otra preciosa información para la ayuda a la navegación aérea tanto civil como militar.

Los datos de mayor interés para la climatología consisten principalmente en los mapas de cobertura del cielo, en los del manto nivoso y en los valores de radiación reflejada por los océanos y por los desiertos, informaciones de las que es posible calcular la diferencia entre la energía global entrante y saliente del sistema Tierra-Atmósfera. Es inútil subrayar que esta diferencia determina la temperatura media de la Tierra y condiciona el clima a escala planetaria.

Las estaciones secundarias SDUS están instaladas, además de la que hay cerca del CNMCA de Roma, en los aeropuertos de Milán, Cagliari, Brindisi y Palermo.

Las estaciones reciben del Meteosat las imágenes analógicas de las transmisiones WEFAX (Weather Facsimile), en las tres bandas de la luz visible, del infrarrojo y del vapor acuoso, según formatos y según un programa de transmisión definidos en el Centro principal de Darmstadt,



Imágenes en falsos colores procesadas en la estación PDUS de Roma.

en Alemania Federal. Las imágenes están constituidas por 800 líneas con 800 pixels por línea y normalmente se utilizan para el análisis del tiempo durante las 24 horas. Estas imágenes, a diferencia de las recibidas a través de la PDUS, están destinadas a una utilización de tipo no numérico.

Cada estación está constituida por un módulo receptor (antena, down-converter, receptor) que capta las señales analógicas enviadas sobre dos canales de telecomunicación en la banda S de 1691.0 y 1694.5 MHz, por un microordenador que regula el flujo de las informaciones y acepta un coloquio interactivo con el usuario, por un sistema de presentación en pantalla en b/n, por un sistema de registro en cinta magnética y por un sistema de reproducción sobre papel fotográfico (en el caso específico, un reproductor fotográfico de rayo láser).

El sistema de mando y control de la estación es muy sencillo y versátil al mismo tiempo, porque el operador puede predisponer el empleo automático de la propia estación.

Las señales recibidas por cada una de las cinco SDUS pueden enviarse por línea telefónica a terminales remotos de tipo facsímil o vídeo (un máximo de diez por cada estación), así como por lo menos cincuenta entidades italianas pue-

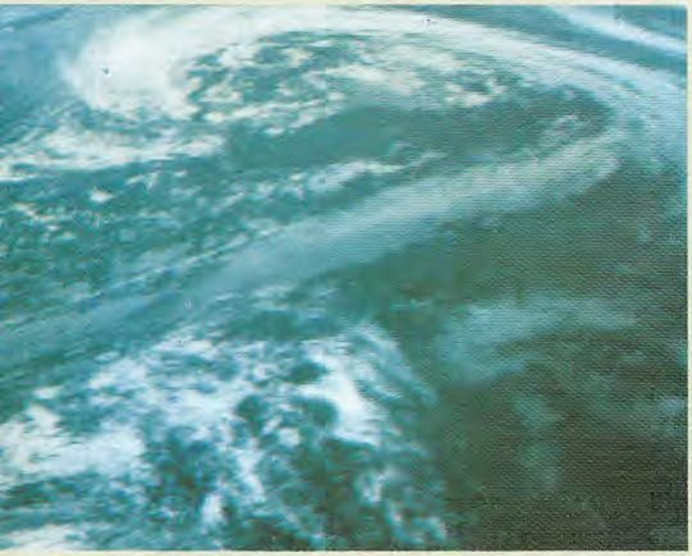


Imagen procesada por una SDUS de un sistema nuboso sobre Europa occidental.

den ponerse fácilmente en condiciones de recibir las imágenes Meteosat en tiempo real.
La posibilidad de registrar las imágenes en cinta magnética (cassette) es un óptimo sistema para constituir un archivo local eficaz, poco voluminoso y relativamente barato.
El sistema de proceso «off-line» ha sido el primer sistema utilizado por el Servizio Meteorologico dell'Aeronautica Militare para estudiar las características de las imágenes del Meteosat y para desarrollar módulos software para el tratamiento de las informaciones digitales de los datos del satélite, programas que una vez puestos a punto se están instalando operativamente en la PDUS.
El sistema puede procesar imágenes recibidas por la estación primaria en tiempo casi real mediante un enlace de alta velocidad y de acceso directo a la memoria de ambos ordenadores.
El sistema está constituido por un ordenador análogo al de la PDUS, con memoria RAM central de 1 Mbyte y memoria masiva en disco de 20 Mbytes. Está dotado de dos unidades de cinta magnética, de un terminal de control, de plotters y presentadores gráficos capaces de visualizaciones en color para poder evidenciar todos los detalles de las imágenes a examinar.
El sistema, llamado ISIDE (Interactive System for Image Data Elaboration), no sólo puede procesar imágenes del Meteosat, sino también de otros satélites, imágenes radar digitales o mapas meteorológicos para superponerlos a las imágenes presentadas en la pantalla. Cada una de las veinte estaciones DCP está dotada de ocho sensores que miden las magnitudes fundamentales en el suelo: intensidad del viento, dirección del viento, presión, temperatura del aire, humedad relativa, precipitación, temperatura en el suelo y radiación solar.
Estas estaciones, completamente automáticas, efectuán una determinación de los parámetros recibidos cada hora, convierten las señales eléctricas de los sensores en parámetros físicos mediante tablas de tarado y codifican las informaciones en el formato previsto.
Los mensajes se registran localmente y se transmiten al Meteosat, el cual reenvía el mensaje de cada DCP al centro operativo de Darmstadt, que lo introduce en la Red Mundial de Telecomunicaciones Meteorológicas.
Las estaciones también están conectadas al CNMCA de Roma mediante líneas telefónicas conmutadas que permiten la obtención de los mensajes en el caso en que se produzca una avería en el satélite.
El nuevo Centro del Servizio Meteorologico para el tratamiento de datos de satélite deberá completarse con los aparatos destinados a la recepción de datos de imagen y digitales enviados a la Tierra por satélites polares TIROS-N.
Como estos últimos tienen una resolución espacial del orden de 1,2 km (cerca de 4 veces mayor que la del Meteosat), proporcionan unas imágenes de calidad muy superior. Además, por disponer de un elevado número de sensores radiométricos en la banda del infrarrojo y en la de microondas, pueden efectuar sondeos de los diferentes perfiles verticales de temperatura con una resolución espacial de aproximadamente 50 km.
En la realización del Centro descrito han contribuido DATAMAT, una firma italiana de ingeniería de sistemas, que ha proporcionado la estación primaria PDUS, las estaciones secundarias SDUS y el sistema «off-line». Otra firma italiana de aparatos de precisión, la SIAP, ha suministrado las estaciones automáticas DCP.
El Servizio Meteorologico dell'Aeronautica Militare utiliza el sistema de satélite de forma continuada (24 horas sobre 24 horas), y los primeros resultados indican que las metas prefijadas pronto se conseguirán plenamente.

De informaciones del Servizio Meteorologico dell'Aeronautica Militare y de Datamat Ingegneria dei Sistemi S.p.A.

Es decir, un campo de intercambio en el programa principal y su correspondiente en la subrutina pueden tener descripciones diferentes, siempre que queden inalteradas la longitud global y el tipo de USAGE utilizada.
Si en el programa llamador se define un campo de la forma

```
01 A.
   05 A1      PIC X(2).
   05 A2      PIC X(4).
```

en la LINKAGE SECTION del programa llamado, el mismo campo puede describirse así:

```
01 B.
   05 B1      PIC X.
   05 B2      PIC X(3).
   05 B3      PIC X(2).
```

Como puede verse, el campo A y el campo B, a pesar de estar subdefinidos de manera diferente, tienen la misma longitud global.
La definición de los campos en la LINKAGE SECTION no comporta la reubicación de la memoria correspondiente, puesto que el subprograma prevé que los campos ya estén definidos en la memoria del programa llamador.
El campo B del ejemplo anterior, definido en la LINKAGE SECTION del programa llamado, ocupa físicamente la misma área de memoria que el campo A definido en la WORKING-STORAGE SECTION del programa llamador. En sustancia, el campo de LINKAGE SECTION de una subrutina puede compararse a una PREDEFINES del campo de WORKING-STORAGE SECTION del programa llamador.

El verbo CALL.

La llamada de una subrutina se efectúa mediante la instrucción CALL (llama), cuyo formato se ha representado en la primera tabla de abajo, donde «literal» representa una cadena de caracteres, encerrados entre comillas, que corresponden al nombre definido en la cláusula PROGRAM-ID del programa llamado; «nombre-de-dato-1, nombre-de-dato-2...» son los nombres de los campos de tránsito definidos en la WORKING-STORAGE SECTION.
El nombre del programa llamado debe ser único: un programa no puede llamar dos subprogramas diferentes con igual PROGRAM-ID.

PROCEDURE DIVISION de una subrutina

Para completar la descripción de las modalidades de utilización de un subprograma, debe tenerse en cuenta que, además de la LINKAGE SECTION, es necesario insertar la cláusula USING en la instrucción PROCEDURE DIVISION según el formato representado en la segunda tabla de abajo, donde «nombre-da-dato-3, nombre-de-dato-4...» son los nombres de los campos de tránsito definidos en la LINKAGE SECTION.
El orden en que los «nombres-de-dato» están especificados en esta instrucción debe respetar fielmente el orden con el que se han definido los datos en la cláusula USING de la CALL correspondiente. La asociación entre campos de tránsito del programa llamador y los definidos en la LINKAGE SECTION del programa llamado sólo se efectúa a través de esta correspondencia.

La instrucción EXIT PROGRAM

Esta instrucción tiene la función de transferir el control de la subrutina al programa llamador, y tiene el siguiente formato:

EXIT PROGRAM

En un subprograma puede haber más de una

FORMATO DE LA INSTRUCCION CALL

<u>CALL</u> literal <u>USING</u> nombre-de-dato-1 nombre-de-dato-2...

PROCEDURE DIVISION DE UNA SUBRUTINA

<u>PROCEDURE</u> <u>DIVISION</u> <u>USING</u> nombre-de-dato-3 nombre-de-dato-4...

ESTRUCTURA DE UN PROGRAMA LLAMADOR

```
IDENTIFICATION DIVISION
PROGRAM-ID. LLAMADOR.
.
.
.
DATA DIVISION.
.
.
WORKING-STORAGE SECTION.
01  A                                      PIC X(8).
01  B.
    05  B1                                     PIC 9(6).
    05  B2                                     PIC X(10).
.
.
.
*
*
PROCEDURE DIVISION.
MAIN SECTION.
INICIO.
    .
    .
    .
    .
    CALL 'VERIFICA' USING B
                          A.
    .
    .
    .
```

ESTRUCTURA DE UN PROGRAMA LLAMADO

```
IDENTIFICATION DIVISION.
PROGRAM-ID. VERIFICA.
.
.
.
DATA DIVISION.
.
.
WORKING-STORAGE SECTION.
.
.
.
LINKAGE SECTION.
01  C                                      PIC X(8)
01  D.
    05  D1                                 PIC 9(6).
    05  D2                                 PIC X(4).
    05  D3                                 PIC X(6).
*
*
*
PROCEDURE DIVISION USING D
                         C.
PRIMERA SECTION.
INICIO.
    .
    .
    .
    .
    .
    .
SALIDA.
    EXIT PROGRAM.
```



Operación de carga de una bobina en la unidad de cinta.

EXIT PROGRAM, en función de las diferentes vías de salida previstas.
La estructura esquemática de un programa principal (LLAMADOR) que usa una subrutina de PROGRAM-ID VERIFICA se ha representado en el primer listado de la página anterior, mientras que la estructura del subprograma VERIFICA puede verse en el segundo listado. Observando los detalles indicados puede verse la aplicación de algunas de las reglas enunciadas:

— los campos A y B, descritos en la WORKING-STORAGE SECTION del programa llamador (LLAMADOR) se han pasado al subprograma mediante la instrucción CALL VERIFICA' USING B A
— el LITERAL usado en la instrucción de llamada contiene el PROGRAM-ID de la subrutina (VERIFICA)
— mediante la línea simbólica de la subrutina PROCEDURE DIVISION USING D C' se establece la correspondencia entre los campos del programa llamador y los del programa llamado.

Obsérvese el orden de especificación de los campos: tanto en la instrucción de llamada

CALL 'VERIFICA' USING B A

como en la

PROCEDURE DIVISION USING D C

primero se referencia el campo de 16 caracteres y después del de 8 caracteres.

Ejemplos de aplicación

Al final de este capítulo presentamos dos ejemplos de subrutinas de uso muy frecuente:

1 / CONTRFECHA, para el control y la conversión de una fecha
2 / CALENDARIO, para el cálculo del día de la semana que corresponde a una fecha.

Las funciones realizadas por las dos subrutinas se explican a nivel de REMARKS.
Obsérvese que la rutina CALENDARIO, antes de efectuar el cálculo requerido, llama la CONTRFECHA pidiéndole el control de la fecha recibida por el programa principal.

EJEMPLO DE APLICACION: CONTROL DE UNA FECHA

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CONTRFECHA.
 REMARKS. ==========================================================
          = SUBRUTINA PARA EL CONTROL Y CONVERSION DE LA FECHA =
          ==========================================================
             LA CONVERSION SE REALIZA SOBRE LA BASE DEL VALOR
             DEL CAMPO OPCION:
             OPCION = 1
             CONTROL DE LA FECHA EN EL FORMATO DDMMAA
             OPCION = 2
             CONTROL DE LA FECHA EN EL FORMATO DDMMAA Y CONVERSION
             AL FORMATO AAMMDD
             OPCION = 3
             CONTROL DE LA FECHA EN EL FORMATO DDMMAA Y CONVERSION
             AL FORMATO JULIANO (AADD)
                                 = = =
             LA FECHA, CONTROLADA Y EVENTUALMENTE CONVERTIDA, SE
             RESTITUYE EN EL CAMPO FECHA-INOUT
             SI LA FECHA DADA RESULTA CORRECTA, EN EL CAMPO ERROR
             SE RESTITUYE EL VALOR, SI NO EL VALOR 1.
*
*
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. .............
 OBJECT-COMPUTER. .............
*
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  AUXILIAR-DDMMAA.
     05  DIA.
         10  DD                      PIC 99.
     05  MES.
         10  MM                      PIC 99.
     05  AÑO.
         10  AA                      PIC 99.
*
 01  AUXILIAR-JULIANA  REDEFINES  AUXILIAR-DDMMAA.
     05  FILLER-9                    PIC 9.
     05  AA-JULIANA                  PIC 9 (2).
     05  DD-JULIANA                  PIC 9 (3).
*
 01  AUX-99                          PIC 99.
*
 01  RESTO                           PIC 9 COMP.
     88  BISIESTO  VALUE 0.
*
*
*            TABLAS
*
```

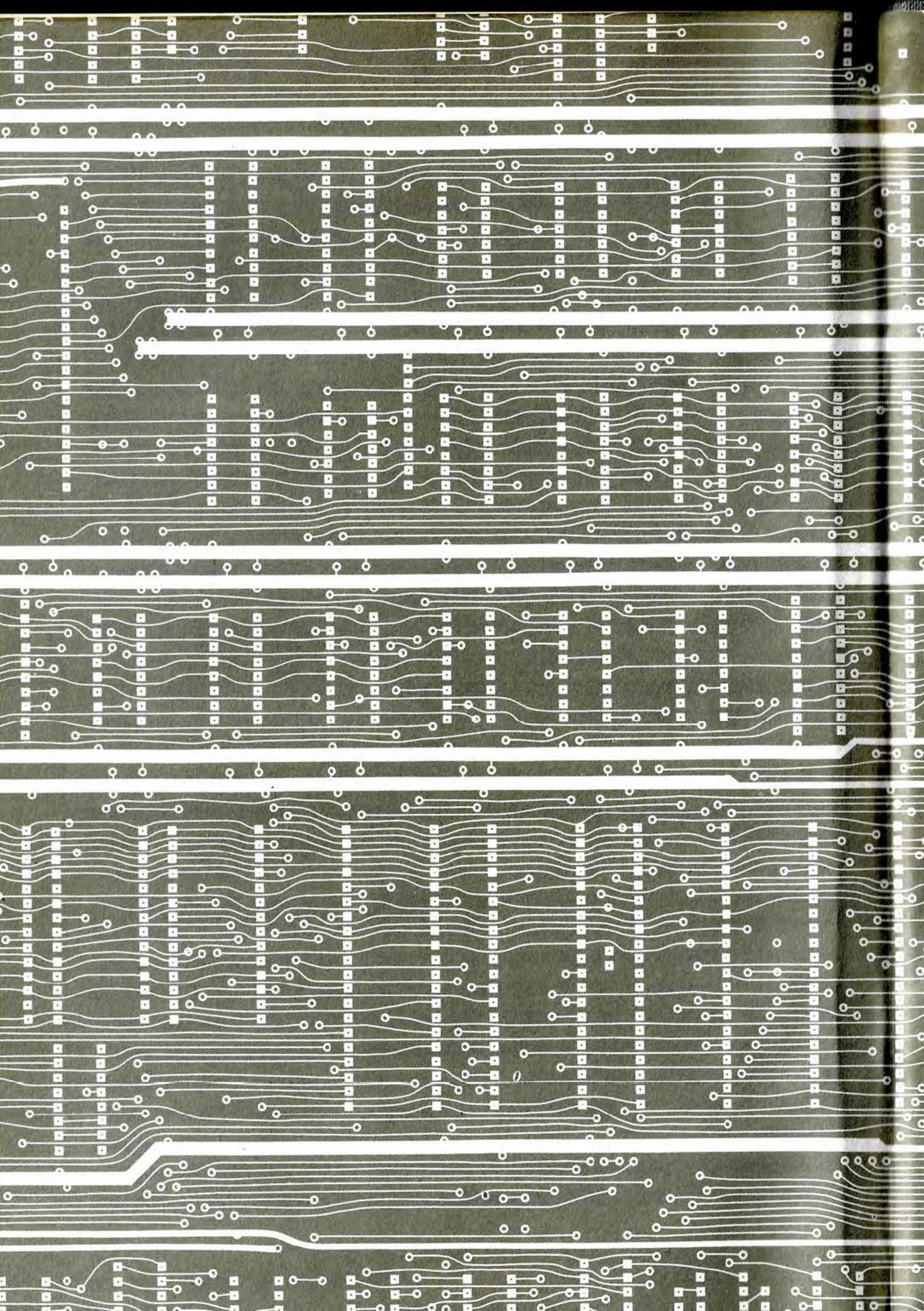
```
*
 01  DIAS-X-MES.
     05  FILLER                        PIC X(24) VALUE
         '312831303130313130313031'.
 01  TABLA-DIAS  REDEFINES  DIAS-X-MES.
     05  NUM-DIAS                      PIC 99 OCCURS 12.
*
 01  BASES-JULIANAS.
     05  FILLER                        PIC X(36) VALUE
         '000031059090120151181212243273304334'.
 01  TABLAS-BASES  REDEFINES  BASES-JULIANAS.
     05  BASE                          PIC 9(3) OCCURS 12.
*
*
*
 LINKAGE SECTION.
 01  OPCION                            PIC 9.
 01  FECHA-INOUT.
     05  DD-INOUT                      PIC 99.
     05  MM-INOUT                      PIC 99.
     05  AA-INOUT                      PIC 99.
 01  ERROR                             PIC 9.
*
*
*
 PROCEDURE DIVISION USING OPCION.
                          FECHA-INOUT
                          ERROR.
*
 INICIO.
     MOVE 0 TO ERROR.
 CONTROL-OPC.
     IF  OPCION NOT = 1
                AND = 2
                AND = 3
         MOVE 1 TO ERROR
         DISPLAY '***OPCION'
                 OPCION
                 ' NO PREVISTA ***'
                 UPON PRINTER
         GO TO EXIT-PROGRAM.
     MOVE FECHA-INOUT TO AUXILIAR-DDMMAA.
 CONTROL-INPUT.
     IF AUXILIAR-DDMMAA = SPACES
        MOVE 1 TO ERROR
        DISPLAY '*** FECHA NO ESPECIFICADA ***'
                UPON PRINTER
        GO TO EXIT-PROGRAM.
     INSPECT DIA REPLACING LEADING SPACES BY ZEROES.
     INSPECT MES REPLACING LEADING SPACES BY ZEROES.
     IF DIA NOT NUMERIC
        OR
```

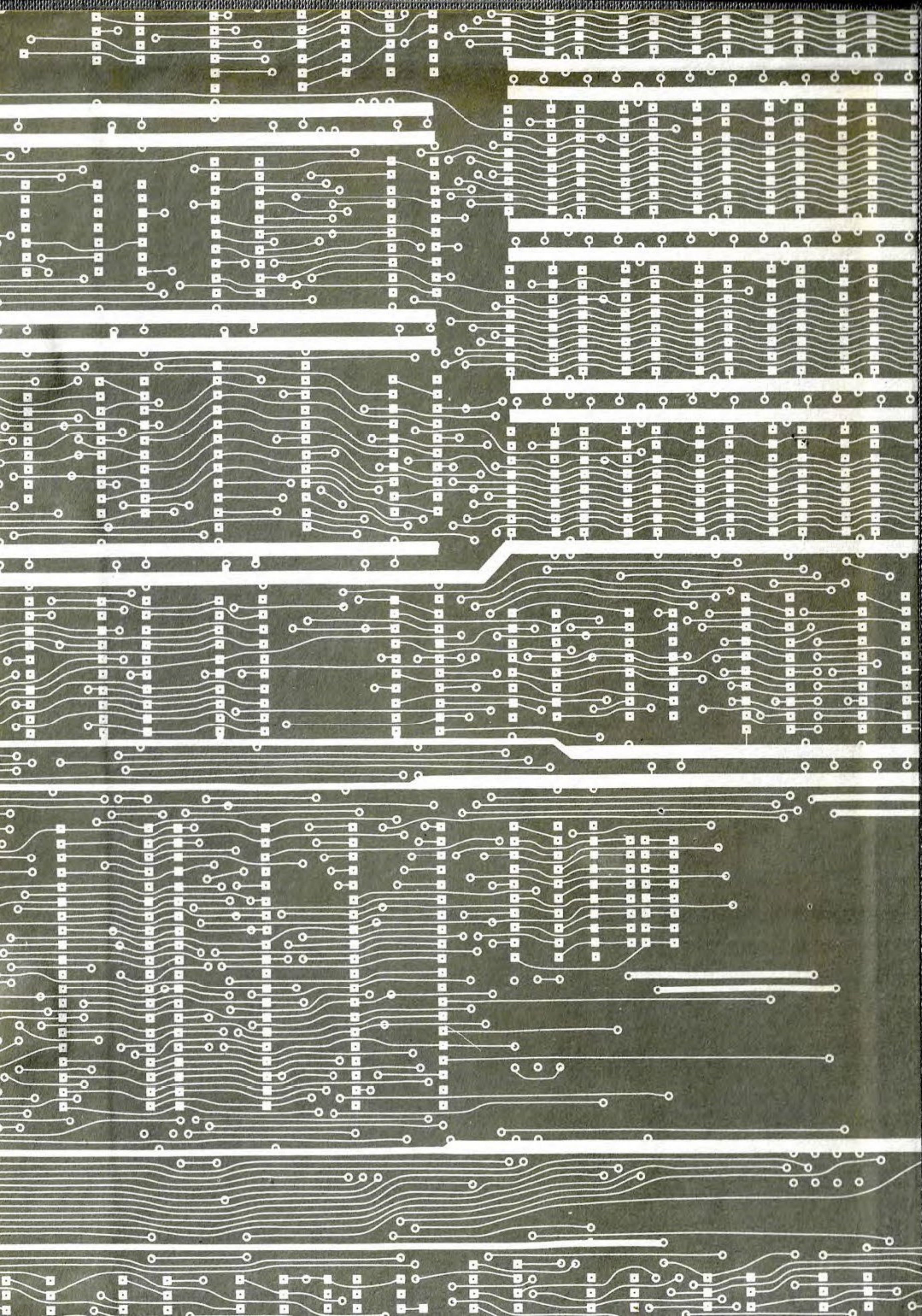
```
           MES     NOT NUMERIC
           OR
           AÑO     NOT NUMERIC
           MOVE 1 TO ERROR
           DISPLAY '*** FECHA NO NUMERICA ***'
                     UPON PRINTER
           GO TO EXIT-PROGRAM.
    CONTROL-MES.
        IF  MM   GREATER THAN 12
           OR
           LESS THAN 1
           MOVE 1 TO ERROR
           DISPLAY '*** NUMERO DE MES ERRONEO ***'
                     UPON PRINTER
           GO TO EXIT-PROGRAM.
    CONTROL-BISIESTO.
        DIVIDE  AA  BY  4
                     GIVING AUX-99
                     REMAINDER RESTO.
        IF BISIESTO
          MOVE 29 TO NUM-DIAS (2).
    CONTROL-DIAS.
        IF  DD  LESS    THAN 1
               OR
               GREATER THAN NUM-DIAS (MM)
           MOVE 1 TO ERROR
           DISPLAY '*** DIA DEL MES ERRONEO ***'
                     UPON PRINTER
           GO TO EXIT-PROGRAM.
    CONVERSIONES.
        IF  OPCION = 1
           GO TO EXIT-PROGRAM.
        IF  OPCION = 2
           MOVE DD        TO AUX-99
           MOVE AA        TO DD
           MOVE AUX-99    TO AA
           MOVE AUXILIAR-DDMMAA TO FECHA-INOUT
           GO TO EXIT-PROGRAM.
        MOVE 0  TO  FILLER-9
                     DD-JULIANA.
        MOVE AA-INOUT TO AA-JULIANA.
        IF  BISIESTO
           AND
           MM-INOUT GREATER THAN 2
           MOVE 1 TO DD-JULIANA.
        ADD DD-INOUT
           DD-JULIANA ,
           BASE (MM-INOUT)     GIVING DD-JULIANA.
        MOVE AUXILIAR-JULIANA TO FECHA-INOUT.
   *
   *
    EXIT-PROGRAM.
```

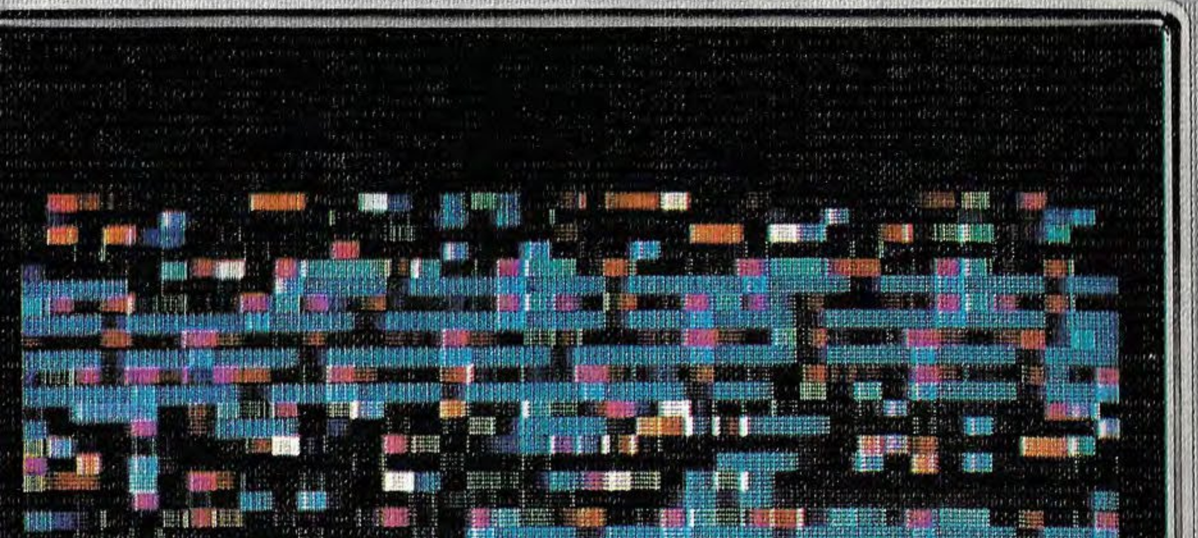
EJEMPLO DE APLICACION: CONVERSION DE UNA FECHA

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CALENDARIO.
 REMARKS. =============================================================
          =                 SUBRUTINA  CALENDARIO                     =
          =============================================================
          LA SUBRUTINA ACEPTA UNA FECHA EN EL FORMATO DDMMAA Y
          RESTITUYE EL CORRESPONDIENTE DIA DE LA SEMANA
          EN EL CAMPO DSET
          PARA CONTROLAR LA FECHA DE ENTRADA Y TRANSFORMARLA EN
          FORMATO JULIANO LLAMA LA SUBRUTINA CONTRFECHA
          EL CALCULO SE EFECTUA ASUMIENDO COMO BASE EL
          DIA 1 ENERO 1900 (LUNES)
                           = = =
          EN CASO DE ERROR SE PARA EL PROCESO.
*
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. .............
OBJECT-COMPUTER. .............
*
*
DATA DIVISION.
WORKING-STORAGE SECTION.
 01   OPCION                           PIC 9.
 01   ERROR                            PIC 9.
 01   AUXILIAR-DDMMAA.
      05  DIA.
          10  DD                       PIC 99.
      05  MES.
          10  MM                       PIC 99.
      05  AÑO.
          10  AA                       PIC 99.
*
 01   AUXILIAR-JULIANA  REDEFINES  AUXILIAR-DDMMAA.
      05  FILLER-9.                    PIC 9.
      05  AA-JULIANA                   PIC 9(2).
      05  DD-JULIANA                   PIC 9(3).
*
 01   COCIENTE                         PIC 9(5).
 01   RESTO                            PIC 9(5).
 01   APOYO                            PIC 9(5).
 01   IND                              PIC 9.
*
*
*               TABLAS
*
*
```

```
 01   TABLA-NOMBRES.
      05  FILLER                         PIC X(3)  VALUE  'DOM'.
      05  FILLER                         PIC X(3)  VALUE  'LUN'.
      05  FILLER                         PIC X(3)  VALUE  'MAR'.
      05  FILLER                         PIC X(3)  VALUE  'MIE'.
      05  FILLER                         PIC X(3)  VALUE  'JUE'.
      05  FILLER                         PIC X(3)  VALUE  'VIE'.
      05  FILLER                         PIC X(3)  VALUE  'SAB'.
*
 01   NOMBRES  REDEFINES  TABLA-NOMBRES.
      05  NOMBRE-DIA                     PIC X(3)  OCCURS 7.
*
*
 LINKAGE SECTION.
 01   FECHA-IN                           PIC X(6).
 01   DSET                               PIC X(3).
*
*
 PROCEDURE DIVISION USING FECHA-IN
                          DSET.
*
 INICIO.
      MOVE FECHA-IN      TO    AUXILIAR-DDMMAA.
      MOVE 3             TO    OPCION.
      CALL 'CONTRFECHA' USING OPCION
                              AUXILIAR-DDMMAA
                              ERROR.
      IF  ERROR NOT = 0
          DISPLAY '*** PROCESO INTERRUMPIDO ***'
                 UPON PRINTER
                 STOP RUN.
*
*
 CALCULO.
      DIVIDE AA-JULIANA BY 4
                        GIVING COCIENTE
                        REMAINDER RESTO.
      IF  RESTO = 0
          SUBSTRACT 1 FROM COCIENTE.
      COMPUTE APOYO = AA-JULIANA * 365 +
                      DD-JULIANA +
                      COCIENTE.
      DIVIDE APOYO BY 7
                   GIVING COCIENTE
                   REMAINDER RESTO.
      COMPUTE IND = RESTO + 1.
      MOVE NOMBRE-DIA (IND) TO DSET.
*
*
 EXIT-PROGRAM.
      EXIT PROGRAM.
```







EDICION
FORUM

4

BASIC

ENCICLOPEDIA DE LA INFORMATICA
MINIORDENADORES Y ORDENADORES PERSONALES